EFFICIENT APPROACHES FOR ROBOTIC ASSEMBLY LINE BALANCING PROBLEMS

JANARDHANAN MUKUND NILAKANTAN (B. TECH)

Submitted in total fulfilment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

SCHOOL OF ENGINEERING

MONASH UNIVERSITY

FEBRUARY 2015

ABSTRACT

JANARDHANAN MUKUND NILAKANTAN (B. TECH)

Assembly Line balancing (ALB) problems deal with the allocation of the tasks among workstations such a way that the precedence relations are not violated and a given objective function is optimized. It is a fundamental problem in continuous production line, and it is one of the difficult optimization problems. Installing assembly line is a long-term decision and required high capital investments. Hence, it is very important to design the assembly line and balance the workload on the workstations. The assembly line has to be rebalanced periodically or if there is a change in the production plan or process. Based on the strategic goals of the manufacturers, the performance measures have to be carefully chosen, since balancing decisions have a long term effect.

Due to the technological advancements, human workforce is replaced by robots to perform the tasks in an assembly line. Different robots with different capacity and specialization are available to perform the assembly task, hence it is required to choose the best fit robot among the available robots such a way that it helps in improving the productivity of the assembly line. Robotic assembly line balancing (RALB) problem aims at assigning the tasks to workstation and allocate robot for each workstation in such a way that the productivity is improved. Very few researchers have proposed models for balancing a robotic assembly line.

The main objective of this research is to develop efficient algorithms to solve robotic assembly line balancing problems. RALB problem is NP-hard, since the basic version of assembly line balancing problems falls under this category. To solve problem of this nature it is necessary to use metaheuristic algorithms. RALB problems with different objective functions are proposed and solved.

The objectives considered for the RALB study are: minimizing cycle time, minimizing energy consumption, minimizing assembly line cost and maximizing line efficiency of a robotic assembly line. Straight and U-shaped RALB problems are considered. The results obtained for the two assembly line problems are compared.

RALB problem with an objective of minimizing cycle time is solved using Particle Swarm Optimization (PSO) and hybrid models of PSO and efficient metaheuristics. Two allocation procedures are used for allocating tasks and robots in the assembly line. PSO and its variants are the metaheuristics proposed to solve the RALB problem. PSO is also hybridized with Genetic Algorithm and Cuckoo search to solve RALB problem. Proposed algorithms are able to produce better results when compared with the benchmark results reported in the literature.

Manufacturing industries give importance to the reduction of energy consumption due to the increase in energy cost and to create an eco-friendly environment. Due to the importance of reducing energy consumption in an assembly line, an energy based RALB problem is proposed. RALB problem with an objective of minimizing energy consumption for straight and U-shaped robotic assembly line is proposed. Particle swarm optimization algorithm is the metaheuristic used to solve the proposed model.

Cost reduction is one of the important tasks for the manufacturing companies throughout the world. In this thesis, a cost based RALB problem is also proposed. The objective considered in this problem is to minimize the assembly line cost. Particle swarm optimization and Differential evolution algorithms are proposed to solve the problem. Straight and U-shaped robotic assembly line problems are solved using the proposed algorithms and the results obtained are presented

Since the investment in assembly line is high, industries try to maximize their usage in the shortest time possible. Maximizing the line efficiency is another measure considered in this thesis. Particle swarm optimization and Differential evolution algorithms are proposed to solve this RALB problem. Line efficiency of both straight and U-shaped configuration of robotic assembly line is compared.

The research on RALB problem optimizing various performance measures considered reveals that U-shaped robotic assembly line is better than straight robotic assembly line.

***Keywords*: Robotic assembly line, Optimization, Metaheuristic Algorithms.**

DECLARATION

I declare that, to the best of my knowledge, the research described herein is original except where the work of others is indicated and acknowledged, and that the thesis has not, in whole, or in part, been submitted for any other degree at this or any other university.

Under the Copyright Act 1968, this thesis must be used only under normal conditions of scholarly fair dealing. In particular, no results or conclusions should be extracted from it, nor should be copied or closely paraphrased, in whole or in part without the written consent of the author. Proper written acknowledgement should be made for any assistance obtained from the thesis.

I certify that I have made all reasonable efforts to secure copyright permissions for third party content included in this thesis and have not knowingly added copyright content to my work without owner's permission.

Signatur 

Date: *5th February 2015*

ACKNOWLEDGEMENTS

I wish to express my special appreciation and thanks to my supervisor Professor S.G. Ponnambalam for his advice during my doctoral research endeavour. He motivated me to remain focused on my research objectives to achieve this goal and showed immense interest in my work by always being available to advise me. His advice on both my research as well as my personal career is priceless and I will hold it close to my heart.

I would like to thank Professor Jawahar and Dr Kanagaraj (TCE, India) for their brilliant comments and suggestions. They gave their most genuine feedback, direction, and assistance whenever I needed them. Let me also thank Professor George Huang (University of Hong Kong) for the opportunity he gave me to work under him as part of my doctoral research. I would like to express my humble gratitude to the administrative and technical staff members of the School of Engineering, who were kind enough to advise and help me in their respective roles.

Let me also thank my best friends Ajith, Akshay, Jose, Paul and Rohit, who, from different corners of the world, shared their thoughts over phone calls, e-mails and Skype conversations and were always available whenever I needed them. I would also like to thank all my PhD colleagues - Nishan, Hasuli, Ajay, Amrutha, Vignesh, Jiten, Kasun, Lakshmi, Altaf, Jason, Shan, Yasir, and Rish - for being there to encourage me during my moments of deep anxiety, while awaiting some positive outcome. Let me also thank my housemates Devangi and Rangika for their support during my stay at Indah Villa. I am indebted to Anirudh for his selfless support in editing my thesis.

At this juncture, I express my deep gratitude to my parents-in-law and my sister-in-law for their constant encouragement and immense support during the course of my research. Let me thank my parents and my brother deeply, for their unconditional trust, timely encouragement and continual prayers. Without them, this research would never have been conducted and this thesis be written.

Finally, I thank my love, Mathangi and our love, Ameya, my wife and our little daughter, for their constant motivation and extra support, which helped me remain perseverant to conduct my research in the most efficient way. Mathangi is a great companion whose love, support, best composure and encouragement helped me remain the most productive throughout this venture of mine.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

SALB	Simple Assembly Line Balancing
JIT	Just in Time
UALB	U-shaped Assembly Line Balancing
FAS	Flexible Assembly Systems
RALB	Robotic Assembly Line Balancing
NP	Non-deterministic Polynomial time
GA	Genetic Algorithm
DE	Differential Evolution
PSO	Particle Swarm Optimization
ABC	Artificial Bee Colony
CS	Cuckoo Search
ACO	Ant Colony Optimization
ALB	Assembly Line Balancing
GALB	General Assembly Line Balancing
RAL	Robotic Assembly Line
MOALB	Multi objective Assembly Line Balancing
MALB	Mixed-model Assembly Line Balancing
COMSOAL	Computer Method of Sequencing Operations for Assembly Lines
moGA	Multi-objective Genetic Algorithm
PSONK	named Particle Swarm Optimization with Negative Knowledge (PSONK)
NSGA	Non-dominated Sorting Genetic Algorithm
WR	Wage Rate Method
WRS	Wage Rate Smoothing-Method
PWWD	Positional Weight Wage Rate Difference Method
PW	Positional Weight Method
GALBPS	General Assembly Line Balancing Problem with Setups
EPC	Electric Power Cost
TOU	Time-of-Use
FFS	Flexible Flow Shop Scheduling

hGA	Hybrid Genetic Algorithm
MOES	Multi-Objective Evolution Strategies
IP	Integer Programming
PW	Positional Weight
TR	Time Ratio
PSO-W	Particle Swarm Optimization with inertia weight
PSO-C	Particle Swarm Optimization with constriction factor
TVIW	Time Varying Inertia Weight
TVAC	Time Varying Acceleration Coefficient
OECD	Organization for Economic Cooperation and Development countries

LIST OF NOTATIONS

Indices

i, j	:	Index of assembly tasks, $i, j = 1,2,\ldots N_a$
h	:	Index of robot types, $h = 1,2,\ldots N_r$
s	:	Index of workstation, $s = 1,2,\ldots N_w$

Parameters

c	:	Cycle time
t_{ih}	:	processing time of task i by robot type h
T_s	:	total execution time for workstation s
$pre(i)$	:	set of immediate predecessors of task i
F	:	Set of tasks
H	:	Set of available robots
S	:	Set of workstations
N_w	:	Number of workstations
N_a	:	Number of tasks
N_r	:	Number of robots
sq	:	Sequence of tasks represents feasible solution
P	:	A set of precedence constraints
v_r^{t+1}		Velocity of particle 'r' at generation 't'
C_0	:	Initial cycle time
E	:	Energy consumption ($E=P.t_{ih}$)
E_0	:	Initial energy consumption of an assembly line
E_s	:	total energy consumption for workstation s
e_{ih}	:	energy consumption of task i by robot h
c_{ih}	:	cost of performing the task i by robot h
P_0	:	initial estimation of assembly line cost
LE	:	Line Efficiency
S_k	:	k^{th} workstation time
P_w	:	Positional weight

LIST OF PUBLICATIONS

Journals

Mukund Nilakantan, J., & Ponnambalam, S. G. 2014. Robotic U-shaped Assembly Line Balancing using Particle Swarm Optimization. *Engineering Optimization*, Taylor and Francis. DOI:10.1080/0305215X.2014.998664.

Mukund Nilakantan, J., Huang, G.Q & Ponnambalam, S.G. 2014. An investigation on minimizing cycle time and total energy consumption in Robotic Assembly Line Systems. *Journal of Cleaner Production*, Elsevier, DOI: 10.1016/j.jclepro.2014.11.041.

Mukund Nilakantan, J., Ponnambalam, S.G., Jawahar & N., Kanagaraj, G. 2015. Bio-inspired search algorithms to solve robotic assembly line balancing problems. *Neural Computing and Applications*, Springer. DOI: 10.1007/s00521-014-1811-x

Mukund Nilakantan, J., Ponnambalam, S.G. & Jawahar N. Maximizing line efficiency of a U-shaped robotic assembly line by minimizing the total energy consumption using evolutionary algorithms; Accepted for publication in *Engineering Computations with minor amendments*, Emerald (under review).

Mukund Nilakantan, J., & Ponnambalam, S.G. Solving Cost-based Robotic Assembly Line Balancing Problems for Straight & U-Shaped Configuration; *Submitted to Journal of Manufacturing Systems*, Elsevier (under review).

Conferences

Mukund Nilakantan, J., & Ponnambalam, S. G. 2012. An efficient PSO for type II robotic assembly line balancing problem. *In: Proceedings of IEEE International Conference on Automation Science and Engineering (CASE)*, pp. 600-605. ISBN: 978-1-4673-0429-0.

Mukund Nilakantan, J., & Ponnambalam, S. G. 2014. Solving cost based robotic assembly line problems using variants of particle swarm optimization. *In: Proceedings of IEEE International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)*, pp 469-475. ISBN: 978-1-4799-4190-2.

Mukund Nilakantan, J., & Ponnambalam, S. G. 2014. Optimizing the efficiency of Straight and U-shaped Robotic Assembly Lines. *5th Joint International Conference on Swarm, Evolutionary and Memetic Computing (SEMCCO 2014) & Fuzzy and Neural Computing (FANCCO 2014)*; presented and proceedings will be published in Springer LNCS volume.

Mukund Nilakantan, J., & Ponnambalam, S.G. 2015. Minimizing energy consumption in a U-shaped robotic assembly line. Accepted and to be presented in *18th Conference of Process Integration, Modelling and Optimisation for Energy Saving and Pollution Reduction*; proceedings to be published in Chemical Engineering Transactions.

CHAPTER 1

Introduction

1.1 Overview

An assembly process aims at bringing together two or more component parts in order to form a new product. In an assembly process parts are added successively to an assembly until a final finished product is completed. In a product, which is to be assembled is often designated as a job. For each product manufactured, there are different small components which will be required to undergo assembly operations. Assembly operations can be automated or if the components or required quantities are small, assembly operations are executed at individual workstations manually. In case of large products like aircrafts, ships, etc., the products are fixed at a location and workers move from product to product performing the operation to form the final product. Flow-line is the most common assembly line where products are assembled when the product moves from one workstation to the next in the line and at each workstation the operations are performed.

Henry Ford developed the first assembly line in 1913 and used that assembly line for mass production of Ford model T automobiles (Alp, 2004). Henry Ford observed that time taken for the assembly process could be reduced and quality of the product assembled would increase if the assembly process is divided into small individual tasks and these tasks are distributed among a set of operators working on assembly workstations along the line. The operators are required to work on a set of tasks and are not required to work on all tasks. Thus the operators become specialized to perform a particular set of tasks and this helps in increasing the speed of the work and quality of the product also increases due to their expertise.

For improving efficiency of the work, Ford applied certain operating principles to his production line and this led to the development of flow-line technology. Workers and tools are placed in the sequence of operations so that each part travels the minimum distance and a material handling system is used for transferring parts from one workstation to the next in a sequence. The movement of parts in the line at equal

intervals helped improving the quality, increase the production rate and also the production cost (Fernandes, 1992).

Due to the increased demand and changing market needs, production companies face the challenge of upgrading the production systems. Customers today look for products with different product variants with different distinctive features from other products available in the market. There is an increased demand to make the production system more flexible to meet the demands and needs of the market.

Due to the fluctuating customer demand it is difficult for the mass production line to respond quickly. Production lines are designed in such a way that tasks are grouped to workstations in an orderly manner so that line efficiency is maximized. This problem is known as the Simple Assembly Line Balancing (SALB) problems. Simple assembly lines are arranged in a straight line. Many firms nowadays incorporate just in time (JIT) principles and group technology into assembly line production, and these modern assembly lines are often organized as a 'U- shaped line' (Alp, 2004) repetitive and job shop applying JIT principles is beneficial. By implementing JIT concepts one of the major change in an assembly line would be to replace the traditional straight line with U-shaped assembly line (Toksarı et al., 2008). The advantages of the U-shaped assembly line when compared to straight line are reduction in the movements of the operators, improved productivity, flexible workforce planning depending on the demand during the planning phase and better material handling. Demand fluctuation can be easily tackled in U-shaped assembly line since there are more possibilities of grouping tasks into workstations on the U-shaped line (Monden, 1983, Hirano, 1988). Scholl and Klein (1999b) define the U-shaped assembly line balancing (UALB) problem as an extension of the simple assembly line balancing (SALB) problem with respect to the precedence constraints.

Assembly process is considered to be one of the important contributing factors for the product cost. Hence, extensive efforts have been taken for improving the cost effectiveness and efficiency of the assembly operations. Assembly production lines can be manually operated, automated, or of mixed design. Manual assembly is characterized by high labor costs and for automated assembly there is a need of very high investment for the development of dedicated equipment for the process. In both methods, there is always a need for performing the operations at highly efficient

manner. Development of flexible assembly systems (FAS) equipped with assembly robots is the main method used for improving the flexibility in the production system (Owen, 1986). FAS are used to meet the increasing demands from the customers and the manufacturers are required to reduce the time to the market. Robots play an important role in FAS and helps in performing specialized operations in an assembly line. Use of robots helps in achieving the flexibility and automation in an assembly line. When flexible equipment like robot are used for performing assembly tasks, the issue of designing an assembly line is of utmost importance. The major design issue is to assign tasks at the workstations and select the robot (best-fit) which performs these tasks to its fullest potential (Bukchin and Tzur, 2000).

Robots are programmed to perform a wide variety of assembly tasks and application. Advanced technology has helped to develop different robots with different specifications and capabilities. Balancing the workload of the workstation in a robotic assembly line is an important task and it helps to maximize the production rate of the line. Objective of allocating the proper robot for the workstation is very critical for the performance of robotic assembly line. Robotic assembly line balancing (RALB) aims at assigning tasks to each workstation and assignment of robots to the workstations in such a way that productivity is improved (Levitin et al., 2006). To perform tasks in a workstation, robots with specific tooling is developed. Tooling for the robot is attached to the robot at the workstation in order to avoid wastage of time for tool change. Tooling design can be done after balancing the assembly line. Major objectives of balancing a robotic assembly line include: optimal balancing of the assembly line for a given number of workstations or achieve a given production rate and allocate the best-fit robot to each workstation. When a new product is planned for assembly due to the availability of different robot types, robots need to be reassigned. Each robot has different capabilities and specialization to perform various tasks.

In case of manual assembly line, there is a considerable amount of variation in the actual processing time when compared to the standard time estimated for the line balancing. Due to this, achieving a perfect line balance is only of theoretical importance. But in case of robotic assembly line balancing there would be not much of variation from the designed assembly line and task performance times. Any imbalance in the line and idle time at any of the workstation will result in reducing the performance of the system (Gao et al., 2009).

Research on assembly lines has been an important subject of study for many years in the field of combinatorial optimization and operation research. Problem could be simply defined such that for a set of given tasks, each of the tasks are associated with cost information and the workstation where these tasks can be executed. Assignment of tasks to the workstations is a solution for an assembly line problem.

Variety of problems occurs in an assembly line due to the large number of variations and constraints. With regard to workstations, there could be different models: linear flow workstations, U-shaped workstations and parallel workstations performing the tasks simultaneously. With respect to tasks, they can be classified into tasks which can be grouped (related) or tasks which cannot be grouped (unrelated tasks). Most common optimization subjects in assembly lines are: minimize the number of workstations required to execute all the tasks; the maximum processing time that can be assigned to a single workstation (cycle time); amount of time that a station needs to wait to perform its allocated task (idle time).

1.2 Optimization techniques to solve assembly line balancing problems

Assembly line balancing problem falls under the category of non-deterministic polynomial time hard (NP-hard) (Gutjahr and Nemhauser, 1964). ALB problem falls under the category of NP-hard due to the computational complexity of the problem (Karp, 1972). Bin-packing problem is one such problem where there is no precedence relationship and falls under the category of NP-hard in the strong sense (Erel and Sarin, 1998). Therefore, simplest version of ALB also falls under the same category.

Due to the combinatorial nature of the problem, there is always a need to reduce the time taken for computing. There is no single method available for solving all optimization problems efficiently (Rao, 2009). Hence, over the years number of optimization techniques has been developed for solving combinatorial problems. Optimization techniques can be broadly classified into exact methods (conventional methods) and approximate methods (modern heuristics) (Rothlauf, 2011). Exact optimization methods are those which guarantee an optimal solution and for approximate optimization methods there is no guarantee for optimal solution. Exact methods are fast and gives exact solution. Exact methods are based on exhaustive search and this is only possible when then number of solution is small so that all solution

possibility can be checked within an acceptable time span. Optimization methods like branch and bound and dynamic programming methods work on partially available solutions and it helps in cutting off the parts of search space without evaluating them. These algorithms are often time consuming and they are not used for solving real-world engineering application problems (Sivanandam et al. 2007). These methods cannot be used to solve real-world problems either due to large search space. Other search methods like local search methods and gradient based methods are different. For local search technique, a new point is created within the neighborhood of the current point and if the neighborhood point is better than the current point based on the quality of the solution, it becomes the new current point. Gradient based methods are those types of methods which can be used to solve continuous problems. But most of the real world industrial optimization problems are not continuous instead they are discrete and combinatorial problems.

Approximate methods help at escaping the local optima and try to find the global optimum solution. Advantage of using approximate algorithms is that they are not attached to any specific domain problem. Hence, heuristic methods are used to solve real optimization problems which are generally complex (Martí and Reinelt, 2011). Heuristics tries to produce acceptable solution for a complex problem in reasonable computational time using common sense logic. Size of the problem also sometimes makes it almost impossible to solve optimally. Therefore heuristic methods helps to save computational time but at the cost of not guaranteeing the optimal solution. Metaheuristics are problem-independent techniques and are designed to solve approximately wide range of optimization problems without having to adapt deeply into each problem. The Greek prefix “meta” present in the name is to indicate that these are “higher level” heuristics when compared to problem specific heuristics (Boussaïd et al., 2013).

These algorithms are applied when there are no satisfactory problem specific algorithms to solve them. Complex problems in industries ranging from finance to production management are solved using metaheuristics (Glover and Kochenberger, 2003). The metaheuristics approach for solving optimization problem starts with an obtained initial solution or with a set of initial solutions, and then an improved search guided by certain principles.

Metaheuristic algorithms are developed based on the learning from nature system and are often bio-inspired, and they are widely used algorithms for optimization problems (Yang and Deb, 2014). Nature-inspired (bio-inspired) algorithms are those algorithms which are inspired by nature phenomenon. Genetic algorithm (GA) and Differential Evolution algorithm (DE) are inspired based on the biological evolutionary process. Algorithms like particle swarm optimization algorithm (PSO), artificial bee colony algorithm (ABC), Cuckoo search (CS) and ant colony optimization algorithm (ACO) are developed based on the behavior of animals. These algorithms have been widely applied in various fields. These algorithms receive wide attention from researchers from various fields of engineering due to easiness in implementation and ability to obtain better solution for hard problems (Boussaïd et al., 2013).

Assembly line balancing problems falls under the category of NP-hard (Scholl and Becker, 2006) and solving these problems optimally by total enumeration is not practical with real-world or large-size problems. Thus researchers shift their focus towards metaheuristics approaches as a popular way to address these hard problems. Metaheuristics are efficient as they are fast and simple to implement. The focus of research thus shifts towards the development of efficient metaheuristics algorithms to solve assembly line balancing problems. Various optimization problems are solved by using metaheuristics which provides a general algorithmic framework (Sörensen and Glover, 2013).

From the literature, it could be observed that researchers use optimization or simulation models to solve assembly line balancing problems. In this section the concept of assembly line with straight shaped and U-shaped line configurations are discussed. Concepts of manual and automated assembly lines are also discussed along with optimization techniques available to solve these types of problems are discussed. In this research, metaheuristic algorithms are proposed to solve robotic assembly line balancing problems with different objectives.

Metaheuristic algorithms like particle swarm optimization, differential evolution and hybrid algorithms is to be implemented to solve robotic assembly line balancing problems. These metaheuristic algorithms are used due to the following advantages: fast convergence, fewer parameters setting, and the easiness to implement. Therefore, PSO and DE has been applied to solve different types of engineering problems (Wu et

al., 2011). Hybridized metaheuristics are also implemented and hybrid metaheuristics helps to improve the search capability of algorithms. Hybrid algorithms combines the advantages of each algorithm, while minimizing any significant disadvantage. Hybridization can generally make some improvements in terms of quality of the solution and the computational time (Ting et al., 2015).

Robotic assembly line with straight line configuration to minimize energy consumption and assembly line cost has not been addressed by earlier researchers. Mathematical models for Robotic U-shaped balancing problems to optimize cycle time, energy consumption and assembly line cost has not been addressed by earlier researchers. The objective of this research is to focus on these areas.

1.3 Organization of the thesis

In this thesis an attempt has been made to develop metaheuristic algorithms for solving robotic assembly line balancing problems. This research also aims at proposing mathematical models to solve robotic assembly line balancing problems with different objectives. The chapters in this thesis are organized in the following manner.

Chapter 2 **-** *Literature Survey* **-** The detailed literature survey on different types of assembly line balancing problems are presented in this chapter. This chapter also gives the details of different optimization techniques used by researchers to solve assembly line balancing problems. The research gaps and a summarized state of the art is also presented based on the detailed literature survey.

Chapter 3 **-** *Problem Definition* **-** Details of research problems considered in the present work is discussed via problem statement, research objectives and research approach.

Chapter 4 **-** *Mathematical Models for RALB problems* **-** The details of mathematical models for RALB problems with different objectives are presented. Assumptions for the problems considered are also presented.

Chapter 5 **-** *PSO and Hybrid PSO for RALB Problem to minimize cycle time* **-** Different metaheuristics are applied to solve RALB problems with the objective of minimizing the cycle time. Details of the experimental study and case studies are presented here. Parameters for different metaheuristics are investigated through

experiments. The parameter selection procedure is explained. Solutions obtained using different metaheuristics are presented in detail. Comparative study is conducted for those problems where benchmark results are available. RALB problems with to two configurations (straight and U-shaped) are discussed in detail.

Chapter 6 – *PSO to solve energy based RALB problems*- This chapter presents RALB problems with an objective of minimizing energy consumption in a robotic assembly line. Metaheuristics are proposed to solve the problem. Results obtained for the proposed models are presented in this chapter.

Chapter 7 - *PSO and DE to solve cost based RALB problems* - Detailed performance evaluation of the proposed algorithm to solve the RALB problem with an objective of minimizing total assembly line cost for straight and U-shaped robotic assembly line are presented in this chapter.

Chapter 8 - *PSO and DE for RALB problem to maximize line efficiency* - This chapter presents different approaches adapted to solve the RALB problem with an objective of maximizing line efficiency for straight and U-shaped robotic assembly line.

Chapter 9 - *Conclusion* - A discussion and summary of the main findings of this research are presented in this chapter. In addition, it also contains the direction in which future work can be carried out.

CHAPTER 2

Literature Survey

This chapter introduces the basic concepts and assesses the current status of research in assembly line balancing (ALB) problems. This section provides details of classification of ALB problems. Furthermore, this section presents the background work of assembly line balancing and robotic assembly line balancing problems which form the basis of this research work. Finally, this chapter also provides overview of solution techniques for solving different variety of ALB problems that have been considered in research studies till date.

2.1 Assembly lines

An assembly line is a manufacturing process where parts of a product are combined in accordance with a predetermined sequence. The basic form of an assembly line consists of a set of workstations, connected to each other through transportation mechanisms, usually conveyor belts. In order to produce or manufacture a type of product set of tasks is repeated at each workstation. Global market continuously gives pressure to manufacturer to compete with competitors from all over the world due to increased market demand. Hence, manufacturer needs to speed up the time to market and should try to minimize the cost of production for remaining competitive in the market (Alp, 2004). In manufacturing sector, assembly is considered one of the important processes. Assembly process account for more than 20% of total manufacturing cost and consumes up to 50% of total production time (Pan, 2005). Assembly lines are mostly used for car manufacturing, electronic appliances and computer assemblies.

Modern assembly line and its basic concept are credited to Ransom Olds (Domm, 2009). Olds used the concepts to build the first mass produced automobile in 1901. Henry Ford in 1913 modified the assembly line by introducing conveyor belts where they could produce a Model T in ninety three minutes often overshadows the development of Olds (Capacho Betancourt, 2008). Assembly lines are most commonly found in automotive industries and other industries where assembly of washing machines, mobile phones, refrigerators and computers are considered. In the recent

years assembly lines are used for low volume production of customized products (Scholl et al., 2008).

Over the years due to the demand different types of assembly lines based on the requirement have been developed. Due to these developments in assembly lines, need for balancing the assembly lines arises. Design of efficient assembly lines received considerable attention from both companies and academicians over the years. A well-known assembly design problem is assembly line balancing (ALB), which deals with the allocation of the tasks among workstations so that a given objective function is optimized. Assembly line balancing is defined as follows by Erel et al. (2001): Line balancing is the process of allocating a set of tasks to an ordered sequence of stations in such a way that performance measures like cycle time, number of stations are optimized subject to precedence relations among the tasks. Classification of assembly line balancing problems is presented in Section 2.3.

2.2 Definition of assembly line balancing (ALB)

Assembly line consists of set of workstations along a conveyor belt or any material handling mechanism which is capable of moving one piece from one workstation to another. The piece/objects enter the assembly line and moves from one workstation to another workstation till the end of the line. Tasks (operations) are repeatedly performed on the piece which enters a workstation; the time between two pieces which enters the workstation is named cycle time. Work is divided into elementary units named tasks. These tasks are not further divisible and the time taken to perform the task is task time or processing time. These tasks are subjected to restrictions like precedence constraints. The aim of balancing is to allocate equivalent amount tasks to different workstations in an optimal way and reduce the cycle time.

Different industrial environments use assembly line production systems for manufacturing a large variety of products. Consumer good like cars, engines, domestic appliances and other electrical appliances are assembled in an assembly line. Products are different and so it is necessary to implement different production systems (Scholl and Klein, 1999a). An existing assembly line is to be re-balanced regularly or after changes in the present production plan. Objective functions need to be carefully chosen

because of the long-term effect of the balancing decisions, keeping in mind the strategic goal of the manufacturer (Gen et al., 2008).

2.2.1 Basic terminologies

- **Tasks (operations)**: Assembling a product on a line requires dividing the total work content into a set of elementary operation. Task is the smallest, indivisible work element of the total work content.
- **Task time**: The time required to perform the smallest work element during the assembly process.
- **Cycle time**: The time interval between the completion times of two consecutive units.
- **Workstation**: Area in a workplace which is equipped with operators/robots to perform the tasks
- **Precedence relations**: The predetermined order in which tasks needs to be assembled. A task cannot be processed if any of its predecessors is not processed.
- **Workstation time**: The sum of task times of the task allotted in the workstation.

2.3 Classification of ALB problems

There are different kinds of problems in the ALB. Based on Baybars (1986) classification, ALB problems are divided into: the Simple Assembly Line Balancing (SALB) problems and General Assembly Line Balancing (GALB) problems. Based on Ghosh and Gagnon (1989) classification, the two main problems are further classified based on their stochastic and deterministic nature. Figure 2.1 shows the most common classification for assembly line balancing problems.

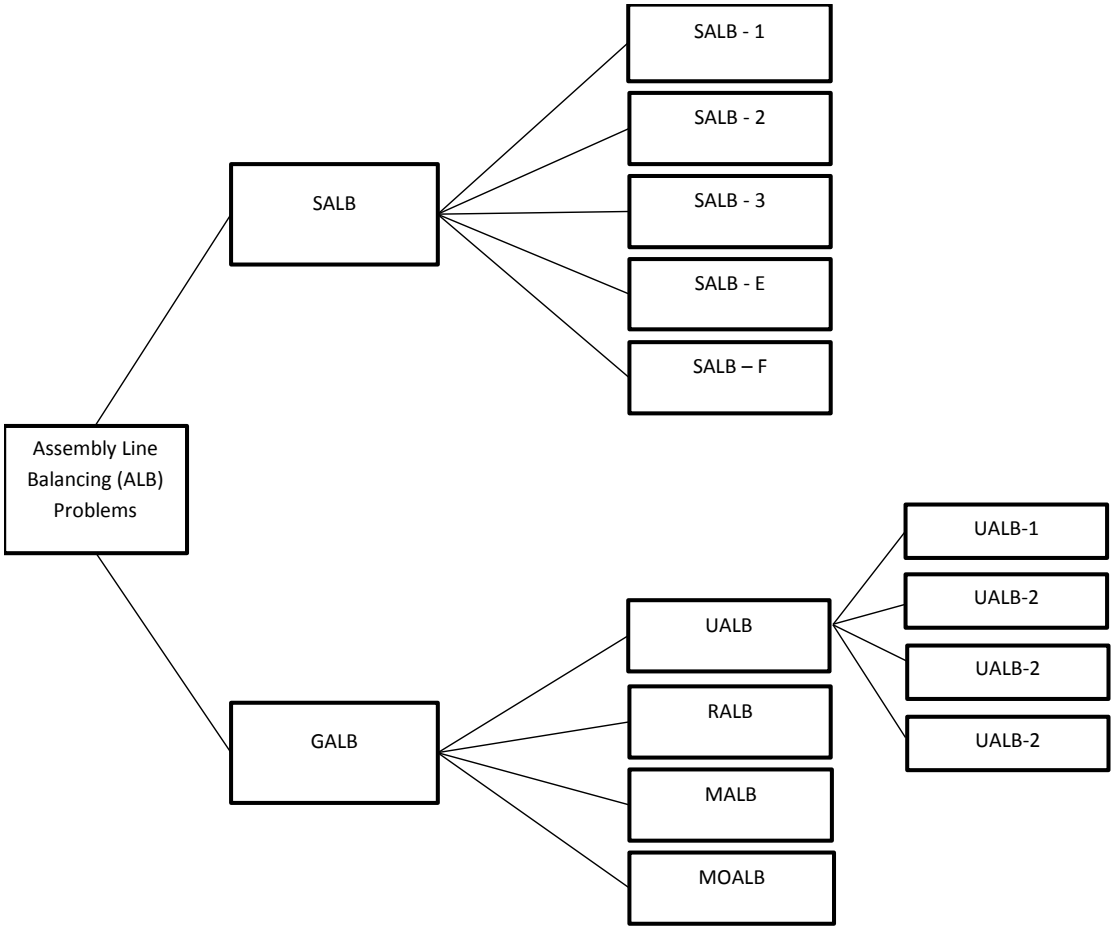

Figure 2.1 Classification of Assembly Line Balancing Problems

2.3.1 Simple Assembly Line Balancing

The basic version of the ALB model is the simple assembly line balancing (SALB) problem. This assembly is capable of producing one type of a product. Different versions of SALB models have been considered (Scholl and Klein, 1999a). SALB can be classified by its objective function and their different problem versions are SALB - 1, SALB-2, SALB-F and SALB-E (Kilincci and Bayhan, 2006). SALB-1 aims at assigning tasks to workstations such a way that number of workstations is minimized whereas SALB-2 problem aims at minimizing the cycle time by assigning the tasks to the given set of work stations. Compared to the previous version, SALB-F determines whether or not a feasible assembly configuration exists for a given combination of cycle time and number of workstations. SALB-E attempts to maximize the line efficiency by minimizing the number of workstations and cycle time simultaneously. SALB-2 is suitable for rebalancing an existing line whereas SALB-1 is more appropriate for designing a new assembly line (Capacho Betancourt, 2008). Scholl et al. (2008)

reported the recent surveys of solution procedures for all the above mentioned problems. Other assembly line balancing problems which are not SALB is classified into a group called General assembly line balancing (GALB). Most problems under GALB are U-shaped assembly line, two-sided assembly line and robotic assembly lines problems.

2.3.2 U-shaped Assembly Line Balancing

In the recent years manufacturers have adopted the principle of just-in time approach for manufacturing processes. This helps to improve the productivity, profit and quality of the product. Just in Time (JIT) is beneficial for the companies which do repetitive jobs. Due to implementation of JIT techniques there is a requirement of replacing straight assembly line with U-shaped assembly line. U-shaped configuration is more flexible due to different possible ways of allocations. In case of the U-shaped assembly lines entrance and exit are in the same position (Toklu and özcan, 2008). The main advantages of the U-shaped assembly line when compared to a straight line are the reduction of the repetitive movement of operators and improvement in productivity. Operators/workers in U-shaped assembly line become more multi-skilled by performing tasks located in different parts of the line. It also helps in developing communication skills and problem solving skills of the workers (Miltenburg, 1998). U-shaped assembly line is highly flexible and changes can be made depending on the demand. U-shaped line also improves the material handling (Toksarı et al., 2008). Task can be assigned to a workstation after all its predecessor or all successors are assigned to an earlier or the same workstation in U-shaped assembly line. This is the distinguishing feature of U-shaped assembly line balancing problems that must allow for the forward and backward assignment of tasks to workstations (Kara, 2008). Problems involving U-shaped assembly line falls under the category of U-shaped assembly line balancing (UALB) problem. Similar to SALB, problems with different objectives for UALB (UALB-1, UALB-2, UALB-E and UALB-F) are available in the literature (Scholl and Klein, 1999b). Most of the characteristics of SALB defined by Baybars (1986) are also valid for UALB.

Scholl and Klein (1999b) describe three problem versions:

- UALBP-1: Minimize the number of stations for a given cycle time.
- UALBP-2: Minimize the cycle time when the number of workstations are fixed.

- UALBP-E: Maximize the line efficiency *E*.

2.3.3 Robotic Assembly Line Balancing

Based on the level of automation of the assembly line it can be divided into two: Manual assembly line and Robotic assembly line. In case of manual lines, the tasks are performed by human operators. Due to the availability of robots which can work 24 hours a day without fatigue, in the recent years, robots are extensively used in assembly lines and these assembly systems are called Robotic Assembly Lines (RAL) (Levitin et al., 2006).

Automation is changing assembly applications significantly. Using robots in assembly line helps to increase the output (productivity) and to reduce costs. Robots are ideal solutions for assembly applications because they are accurate and consistent. They work quickly without tiring or stopping. Quality of the product improves when assembled and manufactured by robots. Robots guarantee precision, consistency and speed in an assembly line. Advances in technology allows development of robots which can assemble nearly anything, no matter how small or unique. Robots help those assemblies which require less human intervention. Robots have different capabilities and efficiencies to perform assembly tasks. Hence it is required to assign the proper robot for each station in a balanced way. An important problem in this context is how assembly lines are managed and how the assembly line is balanced. The robotic assembly line balancing (RALB) problem is based on a distributing the work among the robots with an attempt to balance the whole assembly line. It aims at maximizing the efficiency of the line. With regards to manual assembly line, there is always a variation in actual task performance compared to the standard time estimated for line balancing. In case of manual assembly line balancing, optimal balance is of theoretical importance but whereas the performance of robotic assembly lines depends strictly on the quality of its balance, and on robot assignment (Levitin et al., 2006).. The two types of RALB problems addressed in the literature are: RALB-1, where the objective is to minimize the number of workstations when the cycle time is fixed and RALB-2 which deals with minimizing the cycle time when the number of workstations are fixed (Gao et al., 2009).

2.3.4 ALB problems based on models, layout and number of products

Assembly lines are classified based on the layout and shape of the line, the number of products and models being processed in the line. Based on the model structure, assembly line balancing problems can be classified based on the number of different products which can be produced on the same line. Three types of problems based on the model type are: single-model assembly line balancing, mixed-model assembly line balancing (MALB) problem and multi-model assembly line balancing (Kumar, 2013).

- Single-model assembly line is the classical configuration in which one model of a unique product type is assembled continuously (Figure 2.2). Example: In an automobile industry same models of a car can be manufactured using this type of assembly line.

Figure 2.2 Single-model assembly line

- Multi-Model assembly line involves more than one product produced in batches. Different models with significant difference between each other are processed in this line (Figure 2.3). This model helps to reduce the setup time significantly (Kumar, 2013). Example: In an automobile industry different car models can be manufactured in batches.

Figure 2.3 Multi-model assembly line

- In mixed-model assembly line different types of a product are assembled simultaneously in the line. The production process does not involve setup time since all types of products will require similar type of task to be executed (Figure 2.4). Example: In an automobile industry different car models can be manufactured but not in batch mode.

Figure 2.4 Mixed-model assembly line

Assembly lines can be classified based on the line or shape of the layout. Different types available are: Serial line, Two-sided lines, Parallel lines and U-shaped lines.

- Serial lines: products are processed and assembled through a group of workstations which are arranged in a straight line (Ajenblit and Wainwright, 1998). Figure 2.5 shows the representation of serial assembly line.

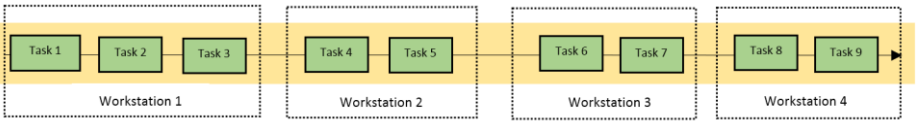

Figure 2.5 Serial assembly line

- Two-sided lines: consists of two serial lines in parallel with pairs of workstations opposite to each other work on the same work piece simultaneously (Figure 2.6). This type of configurations is found commonly in the automotive industry. Left and right sides of the line are used simultaneously to perform different assembly tasks of the same product on both sides (Wu et al., 2008).

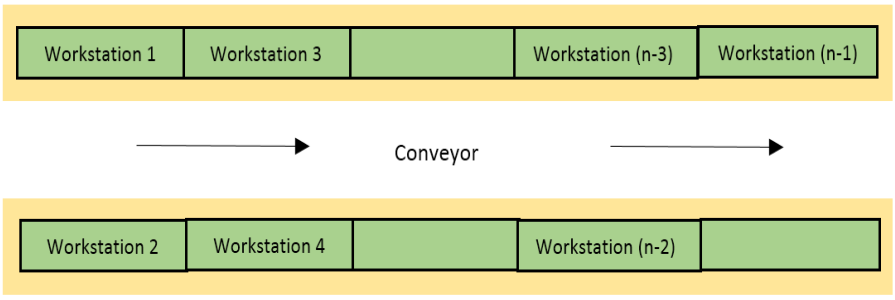

Figure 2.6 Two-sided assembly line

- Parallel lines: this type of assembly lines can be used when multiple products are assembled (Figure 2.7). Same products are also assembled on multiple identical assembly lines based on the demand (Gökçen et al., 2006).

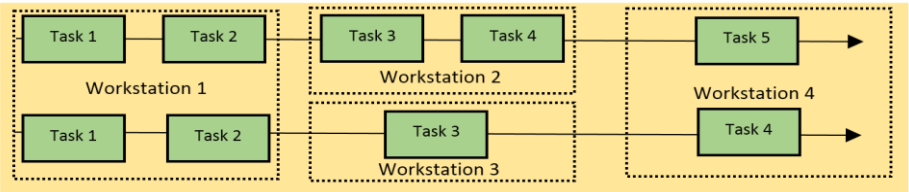

Figure 2.7 Parallel assembly line

- U-shaped line: in this type of layout, tasks are arranged around a U-shaped line. Entry and exit are located in the same side, close to each other (Figure 2.8). This type of layout helps for better management and control (Avikal et al., 2013).

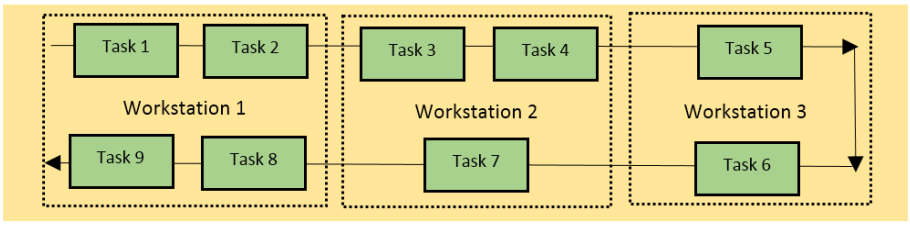

Figure 2.8 U-shaped assembly line

Different researchers have proposed multi-objective assembly line balancing (MOALB) problem where several objectives such as minimizing either the total cost or the number of stations, or maximizing the efficiency of the line, etc. are considered simultaneously (Hamta et al., 2013, Rashid et al., 2012).

2.3.5 ALB problems based on cost and energy consumption

Due to stiff competitive environment, assembly line industries try for attaining the goals of producing products at low cost and high quality in a reasonable time. Manufacturers need to speed up the time to market and at the same time minimize the manufacturing cost to ensure that their products remain competitive (Padrón et al., 2009). Under the economic perspective, cost reduction is considered to be one of the major objectives. Cost-oriented assembly line balancing is a generalized form of time-based assembly line balancing. The cost-oriented assembly line balancing problem aims at assigning all tasks to workstations without violating any precedence relation and by taking into consideration the cycle time production cost is minimized (Rosenberg and Ziegler, 1992). Sarin et al. (1999) considered a stochastic assembly line

balancing problem for the objective of minimizing the total labor cost and the expected incompletion cost arising from tasks not completed within the prescribed cycle time. Amen (2000a) developed a cost-oriented balancing model for a single model assembly line where a vast quantity of one single product is assembled in which the total cost per product unit is minimized. Amen (2000b) dealt with problem which occurs in final assembly of automotive, consumer durables or personal computers where production is very labor-intensive and wage rates are dependent on the requirements and qualification of the workforce. Roshani et al. (2012) mainly dealt with cost-oriented two-sided assembly line balancing problem which occur in the final assembly of products which are very labor intensive.

Extensive studied have been conducted on assembly line balancing problems as seen in the reviews of Becker and Scholl (2006) and Kriengkorakot and Pianthong (2012). Salveson (1955) first formulated mathematically ALB problem. ALB problems mainly deals with assigning tasks to workstations in an assembly line, in such a way that the assignment is in a balanced manner. Classical objective of an assembly line balancing problem is to minimize the number of workstations for a given cycle time. Minimizing the cycle time of the assembly line for a fixed number of workstations is the objective considered and is referred to as time-oriented line balancing (Amen, 2001). Minimizing cost of the assembly line is also an important objective considered in case of assembly line balancing problems. Few minimization objectives considered in case of assembly line balancing problems reported in literature are: throughput time, cost of machinery (Bukchin and Tzur, 2000), inventory cost, labor cost and number of buffers(Capacho Betancourt, 2008). Few maximization objectives considered include production rate, line efficiency and profit (Becker and Scholl, 2006). Recently, Boysen et al. (2007) attempted at classifying the different version of ALB. They proposed an approach which uses tuple notation $[\alpha|\beta|\gamma]$ which is adopted in the classification of machine scheduling. 'α' represents the set of six attributes which determines whether a unique product or model is being considered. 'β' describes the workstation, line considered and it also defines the movement of work pieces, line layout and other constraints. And finally, 'γ' describes the objectives to be evaluated.

Under the economic perspective, minimizing the energy consumption is considered to be one of the major objectives. Extensive efforts are being under taken to improve the efficiency and cost effectiveness of these assembly systems (Sanderson

et al., 1990). One of the major goal of many modern manufacturers in the recent years is to decrease the cost of production by any possible means while satisfying the environmental regulations and ensuring quality (Gungor and Gupta, 1999). Energy consumption is considered to be a very important cost element in a manufacturing enterprise (Kilian, 2008). Due to the rise in energy price and increased demand for environmental compliance, efficient energy management and sustainable energy have become important factors for business competitive advantages. Reduced usage of energy helps the industries to save cost and become more competitive. This is a key factor for promoting green and sustainable practices (Ngai et al., 2013). Significance of reducing energy consumption has been realized in the recent years and stressed more than ever (Liu et al., 2014). Electricity is one of the important forms of energy which is used in a manufacturing sector. Production of electricity is a highly polluted process. Due to the consumption of electricity, amount of carbon dioxide emission generated would be around 20% (Dai et al., 2013). Thus there is a need for manufacturing companies to reduce the energy consumption and become more environmental friendly. Depletion of reserves of energy commodities such as petroleum and other fossils fuels and growing concern over global warming, recently there has been a growing interest for minimization of energy consumption by the industries (Mouzon and Yildirim, 2008). By using energy efficient manufacturing system the energy consumption can be reduced (Chryssolouris, 2005). In manufacturing a car (Press, body, paint and assembly shops) the industry could consume energy up to 700kwh/vehicle. It is reported that energy cost is about 9-12% of the total manufacturing cost and by reducing 20% of the energy cost; the total manufacturing cost can be reduced by 2-2.4% (Fysikopoulos et al., 2012). The authors proposed an empirical study of the energy consumption of an automobile body shop with robot based lines.

It is evident that industrial systems involve a great variety of characteristics and problem variations. Considerable amount of research is being done to fill the gap between research works and real industrial environment. In the literature, there are several models for balancing the different types of assembly line balancing problems. Section 2.3 summarizes the most common assembly line balancing problems.

2.4 Importance of assembly line balancing

Distribution of total workload of the assembly line between each workstations so that idle times are as low as possible is referred to as balancing of an assembly line. Objective of worker/robotic based assembly line is to balance the workload of worker/robot which helps in minimizing the loss and cost. Different components such as products, operations, material handling and assembly line characteristics in the production system need to adapt to any of the changes which occurs to these components without causing much loss. Taking this into consideration, periodically balancing the process, loss for the whole system could be completely removed. Balance losses of an assembly line are bound to happen and there is a very small possibility of obtaining a perfect balance of workload due to dynamic features of the system. In real environment it is not practically possible to allocate work among the workstations equally and hence there is always a balance loss.

By understanding the source of these losses, balance losses can be minimized for the assembly line (Törenli, 2009). An example problem is used to illustrate the losses of balanced and unbalanced assembly lines. Figure 2.9 and Figure 2.10 shows two different balancing conditions of an assembly line for a problem with 4 workstations. The numbers on top of the bars in the graph shows the workstation time. Figure 2.9 shows that workstation 2 is overloaded and work station 3 and 4 needs to wait for workstation 2 to complete the job in the line. Figure 2.10 shows the balanced allocation of workload among the workstations. Idle time at workstation is the primary indicators of the balance losses of an assembly line. Idle time at a workstation shows that there is an excess capacity at that station which is undesired. Idle time of the line is calculated using Equation 2.1.

$$Idle\ time\ of\ \ workstations\ =\ \sum_{i=1}^{N_w}(Cycle\ Time\ of\ \ the\ assembly\ line - workstation\ time) \quad \textbf{(2.1)}$$

In the given example, for unbalanced assembly line the cycle time is 25 and idle time is found out to be 45, whereas in case of balanced assembly line the cycle time is 15 and idle time is found out to be 5. It was analyzed from this example that, the firms will have to balance the assembly line to minimize the loss due to idle time. Balancing the assembly line helps at promoting one piece at a time, avoids overburden at the workstations, helps at minimizing the wastage and reduces the variation.

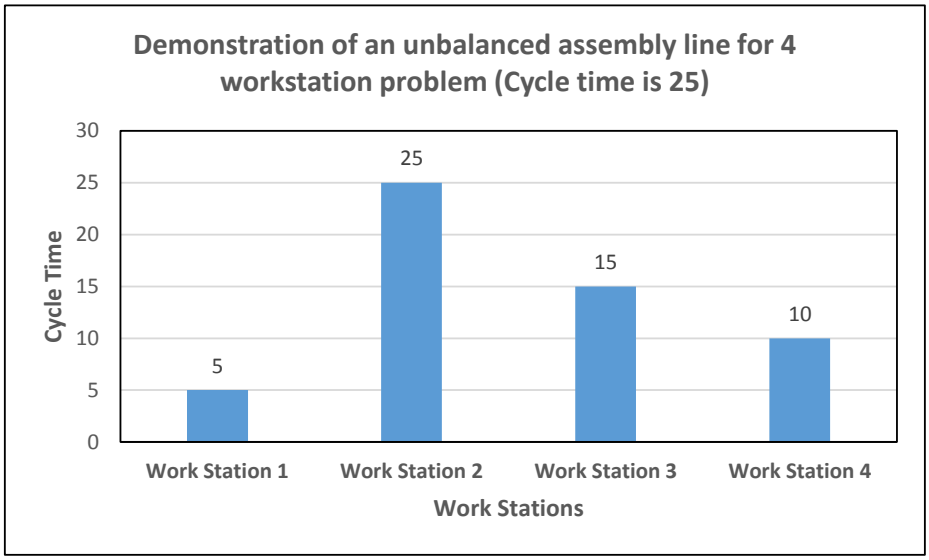

Figure 2.9 Unbalanced Assembly Line

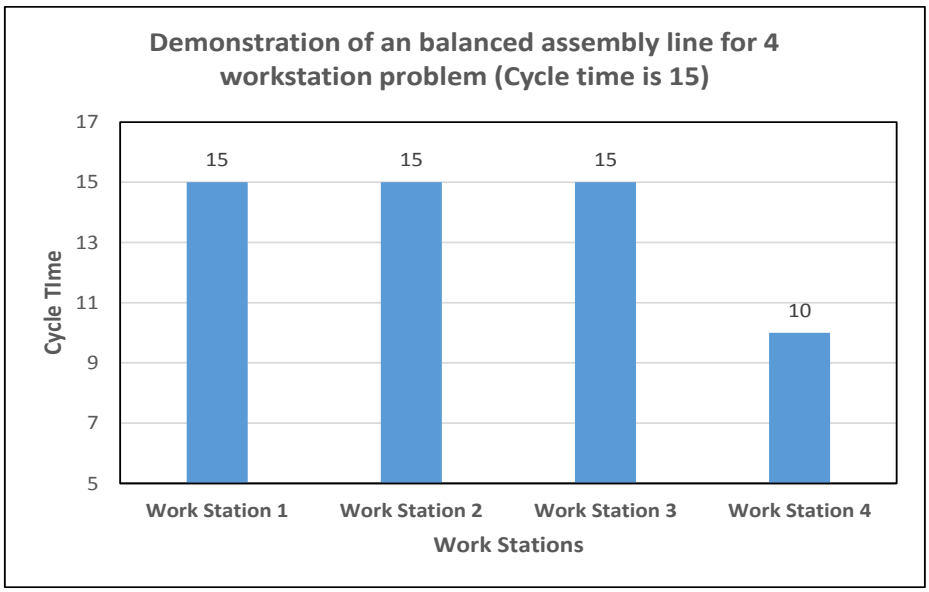

Figure 2.10 Balanced Assembly Line

2.5 Optimization techniques to solve ALB problems

Different types of procedures have been developed to solve assembly line balancing problems. Exact methods have been developed to solve assembly line balancing problems due to the NP-hard nature of the problem. Even though, exact methods guarantee optimum solution, problems with small size assembly tasks could only be solved using these methods due to the high computational load (Capacho Betancourt, 2008). Approximate methods like heuristics and metaheuristics are developed to solve problems with large assembly tasks aimed at producing solutions

which are nearer to the optimal solution. The remaining sections of this chapter gives details of the different procedures reported in literature for solving SALB, UALB and RALB problems.

Few review papers are available which summarizes the different procedures used to solve the assembly line balancing problems. Numerous metaheuristics have been developed to solve SALB problems. A detailed summary of different exact methods and metaheuristics for SALB problems are presented in Scholl and Becker (2006). Becker and Scholl (2006) presented the detailed survey of procedures used for solving problems other than SALB. Detailed classification and overviews of methods used in solving assembly line balancing problems are presented in Sivasankaran and Shahabudeen (2014).

2.5.1 Techniques for solving SALB problems

Details of exact algorithms developed for solving SALB problem are presented in (Baybars, 1986, Erel and Sarin, 1998). Dynamic programming procedure is first developed by Jackson (1956) to solve assembly line balancing problems. It is later modified by (Held et al., 1963). It could be seen that these procedures require large memory requirement. This drawback is reduced by the procedures proposed by Schrage and Baker (1978), Kao and Queyranne (1982). Thangavelu and Shetty (1971) developed a 0-1 integer programming algorithm to solve simple assembly line balancing problem with an objective of minimizing the number of workstations for a given cycle time. Different operational requirements such as zoning, sequencing, idle time, cycle time and cost are considered by Deckro and Rangachari (1990) in developing a goal programming model with an objective of minimizing the number of workstations. Scholl and Klein (1999a) presented the branch and bound algorithm for solving the same problem. For solving SALB-1 effective methods like FABLE, EUREKA, and SALOM are developed (Johnson, 1988, Hoffmann, 1992, Scholl and Klein, 1997). TBB2 and SALOME2 are used to solve SALB-2 problem which uses branch and bound algorithm (Klein and Scholl, 1996). Kilincci and Bayhan (2006) solved SALB-1 problem using Petri-net algorithms. Petri net is a mathematical and graphical tool to model and analyze discrete event systems. Kilincci (2010) solved SALB-2 problem with an objective of minimization of variations in workloads among the workstations. The author developed Petri-net algorithm for solving the problem.

Mathematical models can generate optimal solutions, however in case of large size problems it is difficult to obtain solution in an acceptable time span, hence development of heuristics are required. Ranked positional weight (RPW) proposed by Helgeson and Birnie (1961) is one of the first proposed heuristic. Tasks are ranked in descending order of their positional weights. Other heuristic rules are maximum task time, maximum number of successors, minimum slack, minimum earliest and latest workstation. Arcus (1965) developed COMSOAL (Computer Method of Sequencing Operations for Assembly Lines) to solve SALB problems. A comparative evaluation of six popular assembly line balancing heuristics namely, ranked positional weight, Kilbridge and Wester, Moodie and Young, Hoffman precedence matrix, immediate update first fit, and rank and assign heuristic are presented in (Ponnambalam et al., 1999). The evaluation criteria used are the number of excess stations given, line efficiency, smoothness index and CPU time. Detailed literature survey of heuristic methods can be found in (Erel and Sarin, 1998, Scholl and Becker, 2006). Most research on SALB focuses on SALB-1 and SALB-2, and few studies deal with the optimization of assembly line balancing efficiency (Wei and Chao, 2011). Plans and Corominas (1999) developed a mixed integer programming model and heuristic approach to solve a SALB-E problem. Wei and Chao (2011) proposed a model for SALB-E and the solution procedure is developed which minimizes the total idle time to optimize the assembly line balancing efficiency.

Different types of metaheuristics have been developed to solve assembly line balancing problems. Metaheuristics use different concepts derived from artificial intelligence, evolutionary algorithms inspired from mechanisms of natural evolution (Pierreval et al., 2003). From the literature it could be found that metaheuristics can also be called as soft computing techniques, evolutionary algorithms and nature inspired algorithms. Detailed literature survey of metaheuristics applied on assembly line balancing problems are presented in Rashid et al. (2012) and Sivasankaran and Shahabudeen(2014).

Tabu search (TS) is a metaheuristic algorithm which uses local search methods. These methods are applied to solve SALB-1 and SALB-2 problems (Chiang, 1998, Scholl and Voß, 1997). Ant Colony Optimization (ACO) developed based on the behavior of ants searching for their food. Bautista and Pereira (2002) applied ACO to solve SALB-1 problem. Zheng et al. (2013) proposed an improved version of ant colony

optimization algorithm to solve assembly line balancing (SALB-2) problem. ACO is employed to search for different better combination of tasks to be allotted at each workstation. (Lai and Liu, 2009) uses ACO to optimize the efficiency of a sewing line. Simulated annealing (SA) is a technique inspired from physical annealing of solids has been applied to solve assembly line balancing problems. Suresh and Sahu (1994) applied SA for solving a stochastic variant of SALB-1 problem. Seyed-Alagheband et al. (2011) developed a new simulated annealing algorithm to solve type 2 assembly line balancing with sequence-dependent setup times between tasks. Genetic Algorithm (GA) developed by John Holland follows the biological evolution where the concept of survival of fittest is taken into consideration.

From the literature survey, it could be seen that most of common metaheuristic algorithm which is extensively used in assembly line balancing problems is genetic algorithms and its hybrid versions. Rubinovitz and Levitin (1995) developed a genetic algorithm to solve a single model assembly line balancing problem with deterministic processing time. The results obtained through this method are compared with MUST algorithm (Dar-El and Rubinovitch, 1979). Simple assembly line balancing problem with different objectives: maximizing the number of workstations, minimization of cycle time maximizing the work load smoothness and maximizing work relatedness are solved using a genetic algorithm along with repair method (Kim and Kim, 1996). A multi-objective genetic algorithm for solving simple assembly line balancing problem with objectives of minimizing workstations, maximizing line efficiency and smoothness index is developed by Ponnambalam et al. (2000). A genetic algorithm with heuristic generated initial population is developed by Chong et al. (2008) for solving a simple assembly line balancing problem (SALB-1) with realized cycle time as the fitness function. Performance of this proposed algorithm is compared with results obtained using genetic algorithm with a randomly generated initial population. A hybrid genetic algorithm with a local search is developed by Gonçalves and De Almeida (2002) for solving an assembly line balancing problem with an objective of maximizing the balancing efficiency for a given cycle time. The chromosome representation of the problem in this approach is based on random keys. A heuristic priority rule is used to assign task to the workstations in which the priorities of the operations are defined by the chromosome.

Particle swarm optimization (PSO) developed by Kennedy and Eberhart (1995) is based on the social behavior of bird flocking or fish schooling. PSO is quite similar to GA, but there are no evolution operators in case of PSO. From the literature it could be seen that PSO has been seldom applied for solving simple assembly line balancing problems (Economics, 2011). Economics (2011) applied PSO to solve a SALB-1 problem and the results obtained through this method are compared with results reported in the literature. Petropoulos and Nearchou (2011) applied PSO to solve a simple assembly line balancing problem with two- and three-criteria problem utilizing the cycle time of the line, the workload smoothness among the workstations, and the balance delay time of the assembly line. The results obtained through numerical experiments are compared with the existing two algorithms reported in the literature and it could be seen that PSO performs better in terms of quality of the solution of the problems. Differential evolution (DE) is a metaheuristic developed by Storn and Price (1997). Nearchou (2005) applied DE to solve SALB-1 problem. Extensive experimental work over available benchmarks test problems show the effectiveness of the proposed approach. Later, Nearchou (2007) also applied DE to SALB-2 problem. The results obtained through this method are found to be far superior compared to the results reported in the literature. Several other metaheuristics have been used to solve simple assembly line balancing problems. Detailed literature survey by Rashid et al. (2012) shows the different metaheuristics used to solve different variety of assembly line balancing problems.

2.5.2 Techniques for solving U-shaped ALB problems

In the last decade, issues related to U-shaped assembly lines received more attention (Aase et al., 2004). It could be seen from the literature, research on U-shaped assembly lines are less compared to straight assembly lines. Similar to SALB, U-shaped assembly line balancing can be classified into mainly three groups: UALB-1-objective is to minimize the number of workstations when the cycle time is fixed, UALB-2-objective is to minimize the cycle time when the number of workstations is fixed and UALB-E- objective is to maximize the efficiency of the assembly line. There is a growing interest in the literature to organize traditional assembly lines as U- lines to improve the performance. Although there are many literatures on traditional straight assembly lines, the work on U-shaped assembly line is limited.

Miltenburg and Wijngaard (1994) developed a dynamic programming (DP) formulation for evaluating a single model U-type assembly line with an objective of minimizing the number of stations. The proposed DP formulation could solve small size problems with 11 tasks and for solving large problems they developed a heuristic based on ranked positional weight technique (RPWT). Urban (1998) formulated an integer programing problem to solve a U-shaped assembly line with 45 tasks. Variety of U-shaped assembly line problems is solved by using Branch and Bound method (Scholl and Klein, 1999b). They proposed a technique which is called ULINO (U-line Optimizer) uses a depth-first Branch and Bound algorithm which minimizes the number of stations, cycle time or both. The proposed algorithm is used to obtain optimal results for problems up to 297 tasks. Gökçen and Agˇpak (2006) developed a goal programming model for simple U-shaped assembly line balancing problem for optimizing different objectives by taking into consideration of different constraints like cycle time, assignment, workstation and task load. The proposed model was illustrated with numerical examples. Several reviews on exact methods developed to solve U-shaped assembly line balancing problems are available in literatures (Nakade and Ohno, 1999, Miltenburg, 1998, Zhang and Cheng, 2010).

Like the traditional assembly line balancing problem, U-shaped assembly line balancing problem is also NP hard nature (Miltenburg and Sparling, 1995). Different heuristics and metaheuristics have been proposed to solve U-shaped assembly line balancing problems. Ajenblit and Wainwright (1998) presented a genetic algorithm for UALB with an objective of balancing the workload and minimizing total idle time. Erel et al. (2001) proposed a simulated annealing method for solving a U- type Assembly line balancing problem. Martinez and Duff (2004) proposed heuristic approaches to solve the U-shaped line balancing problem augmented by genetic algorithms. They used ten task assignment rules. Baykasoğlu (2006) developed a new multi-objective simulated annealing (SA) algorithm for solving simple and U type assembly line balancing problems with an objective of maximizing smoothness index and maximizing the efficiency of the assembly line. Task assignment rules are used in constructing feasible solution. The proposed algorithm could obtain optimal results in acceptable time span. Later, Khaw and Ponnambalam (2009) developed a hybrid algorithm by combining 15 task assignment rules and ant colony optimization (ACO) algorithm for solving the same problem. Results obtained through this method are compared with the

results reported in the literature and it could be seen that newly developed algorithm performs better in terms of quality of the solution for most of the problems. Baykasoğlu and Özbakır (2007) developed a multi rule based genetic algorithm (MRGA-SUALB) for probabilistic based U-line balancing problem with the objective of minimizing the number of stations for a given cycle time. The proposed algorithm integrates COMSOAL method, task assignment rules, and genetic algorithm. The results obtained are compared with the optimal solutions and it could be observed that all the problems could achieve the optimal solution except for one problem from the datasets. Hwang et al. (2008) developed a multi-objective genetic algorithm (moGA) to solve the U-shaped assembly line balancing problem. Two performance criteria's (number of workstations (the line efficiency) and the variation of workload) are evaluated. The numerical experiments shows that the proposed algorithm produces good or better results compared to previously reported literature. Sirovetnukul and Chutima (2010) developed a novel algorithm, named Particle Swarm Optimization with Negative Knowledge (PSONK) to solve a single U-shaped assembly line problem aiming to minimize the number of workers, equity of workload and the shortest walking time. The performance of PSONK are compared with Non-dominated Sorting Genetic Algorithm-II (NSGA-II) and it could be analyzed from PSONK outperforms NSGA-II.

Many heuristics and metaheuristics are applied to solve variety of U-shaped assembly line balancing problems and detailed reviews are presented (Kim et al., 2000, Kim et al., 2006). This section presented a summary of most relevant literature related to single model U-shaped assembly line balancing problems with different objectives. Section 2.5.3 presents a summary of procedures used to solve cost and energy based assembly line balancing problems.

2.5.3 Techniques for solving cost and energy based ALB problems

Long term or short terms operating costs are incorporated in cost based line balancing problems. Exact methods, heuristics and metaheuristics have been applied to solve cost based assembly line balancing problems. Researchers considered labor cost, setup cost, equipment cost and inventory cost for developing solutions for cost based assembly line balancing problems (Hazır et al., 2014). The cost based assembly line balancing problem is a generalization of the time based assembly line balancing problem (Rosenberg and Ziegler, 1992). Labor costs contribute significantly to the total

production costs. Labor cost depends on the work content and the qualification of the worker. Due to demand from the market, industries always required to employ new employees, instead companies prefer to go for overtime. However, there will be an increase in cost due to overtime. Rosenberg and Ziegler (1992) developed two new heuristic algorithms (Wage Rate Method (WR) and the Wage Rate Smoothing-Method (WRS)) to solve cost based assembly line balancing where the objective is to minimize the total production cost. The results obtained using the heuristics are compared with the known heuristics Positional Weight Method (PW) and the Positional Weight Wage Rate Difference Method (PWWD). It can be concluded from the experimental results that PWWD and WRS are superior to PW and WR. Amen (2000a) developed a model to minimize the total labor and capital cost. Wage rate of a station is calculated by maximum of wage rates of the allocated tasks, since most demanding tasks define the qualification needed by the worker. An exact backtracking method is developed which is used for solving the cost based assembly line balancing problem. Experimental investigations show that the new method finds optimal solutions for small and medium-size problem in acceptable time. Amen (2000b) developed two new heuristic methods: A new priority rule `best change of idle cost is proposed. This priority rule is different from other priority rules because it is the only one which considers that production cost are the result of both, production time and cost rates. Comparison on the quality of the solution and computational time of the developed algorithm are reported in (Amen, 2001). Scholl and Becker (2005) showed one of the rules developed by Amen is incorrect and gives the corrected and simplified version of this rule. Padrón et al. (2009) developed a combination of heuristic model and exact algorithm with intelligent task allocation or line zone constrains with an objective of minimizing cost solution in a feasible computational time. Cost function considers short term operating costs, task and work station capital investment costs. Erel et al. (2005) considered the probabilistic assembly line balancing problem in U-shaped assembly line with an objective of minimizing total labor cost and total expected incompletion cost for a given cycle time. They used beam search, which is a special type of Tabu Search algorithm. The performance of the proposed method is evaluated on various test problems and the results of the experiments show that the average performance of the proposed method is better than results reported in the literature. Roshani et al. (2012), extended Amen's approach for two-sided assembly lines and used simulated annealing for solving the

problem. Yazdanparast et al. (2011) developed a cost oriented approach and developed a mathematical model to solve General Assembly Line Balancing Problem with Setups (GALBPS). A numerical example is illustrated using Lingo 11 software.

The review reveals that the literature on cost based assembly line balancing problems are very minimal when compared to other types of assembly line balancing problems. However, Hazır et al. (2014) presented a survey paper in which problems, approaches and analytical models on cost based assembly line balancing are analyzed.

Key objectives evaluated in assembly line balancing problems are cost, cycle time and efficiency. It could be seen that research on minimizing energy consumption in manufacturing systems has been rather limited (Dai et al., 2013). Very limited researches related to assembly line problems in the context of minimizing energy consumption are available and few of them are briefly discussed below. Fysikopoulos et al. (2012) presented an empirical study of the energy consumption of an automotive assembly line, under various scenarios and demand profiles are presented by them. (Luo et al., 2013) proposed a multi-objective ant colony optimization metaheuristic to optimize production efficiency and electric power cost (EPC) with the presence of time-of-use (TOU) electricity prices. Dai et al. (2013) proposed an energy efficient model for flexible flow shop scheduling (FFS). A mathematical model for a FFS problem which is based on an energy-efficient mechanism is described by them. An improved genetic-simulated annealing algorithm is adopted due to NP-hard nature of the problem to make a significant trade-off between the make-span and the total energy consumption to implement a feasible scheduling. Mouzon et al. (2007) presented a multi-objective mathematical programming model for scheduling jobs on a single CNC machine with an objective of reducing energy consumption and completion time. Shrouf et al. (2014) developed a mathematical model for minimizing the energy consumption cost in a single machine production system considering variable energy prices during a day. To solve the problem they proposed a genetic algorithm (GA) to obtain 'near' optimal solutions. Performance of the proposed GA is compared with an analytical solution generated. He et al. (2012) developed a model which deals with task oriented energy consumption in a machining manufacturing system by incorporating an event graph methodology. SIMULINK simulation environment is used to solve the model.

This section presented a summary of most relevant literature related to cost and energy based assembly line balancing problems. Section 2.5.4 presents the relevant literature related to robotic assembly line balancing problems.

2.5.4 Techniques for solving RALB problems

Robotic assembly line balancing (RALB) problem is an extension of SALB problems. Robots are replaced to perform the assembly tasks instead of human labor. RALB aims at allocating tasks to the workstations and allocate the best available robot to each workstation. Robotic assembly lines are designed and balanced very well to function efficiently due to high investment required to implement such system. RALB problems are classified into two groups mainly: RALB-1 and RALB-2. This section provides the most relevant literature related to RALB problems.

Graves and Lamar (1983) presented a method for selecting workstations from a set of non-identical candidates and tasks are assigned to the selected workstations. The objective of the method is to maximize the workload of the stations and to minimize the total cost. Graves and Redfield (1988) dealt with a problem where multi-products are assembled, where families of similar products are produced. The objective of the work is to minimize the variable and fixed operating costs of the systems which are capable of producing the products in the desired volumes. To solve the problem they used a graph system where each arc represents the workstation, using the shortest path on the graph, the problem is solved. Khouja et al. (2000) developed a method of using statistical procedures for designing robotic assembly cells. Two stages are there in the proposed methodology. First stage uses a fuzzy clustering algorithm for grouping similar tasks and in the second stage using a Mahalanobis distance procedure appropriate robots are allotted for the task groups. Nicosia et al. (2002) presented a dynamic programming algorithm and introduced several fathoming rules to solve the problem where tasks needs to be assigned to an ordered sequence of non-identical workstations without violating precedence relationships and cycle time. The objective is to minimize the cost of the workstations. The formulation of this work is very similar to RALB problem.

RALB problem is first formulated by Rubinovitz and Bukchin (1991). RALB problem deals at allocating equal amounts of tasks to workstations and assigning the most efficient robots to perform the tasks assigned to the workstations. Objective is to

minimize the number of workstations for a given cycle time. Rubinovitz et al. (1993) developed a RALB problem with the objective of minimizing the number of workstation. Branch and Bond frontier search is used to obtain solutions for very large and complex problems. Bukchin and Tzur (2000) treated the previous problem with a new objective of minimizing the total cost when cycle time is fixed. An optimal and heuristic algorithm is developed for designing a flexible assembly line where several equipment alternatives are available. Small size problems are solved using Branch and Bound algorithms. Large problems are also solved using heuristics proposed. For designing a flexible robotic assembly line with a set of family products, Tsai and Yao (1993) proposed an integer programming model combined with a simulation adjustment phase. The objective of the proposed work is to minimize the standard deviation of the output rates of all workstations which measures the quality of the balance of the line. With the aim of minimizing the total number of robotic cells, Kim and Park (1995) developed a mathematical formulation and a cutting plane algorithm for assigning of assembly tasks on a serial robotic assembly line. The literature mentioned so far reported the works related to RALB-1.

The literature related to RALB-2 discussed in this section mainly deals with the objective to minimize cycle time when the number of workstations are fixed in a robotic assembly line. Levitin et al. (2006) developed a method for the robotic assembly line balancing (RALB) problem with an objective of minimizing cycle time. The method aims at achieving a balanced distribution of tasks amongst the workstations and assigns the best fit robot to perform the tasks allocated to these workstations. Two heuristics methods are proposed for assigning tasks and robots. Genetic Algorithm (GA) is proposed to solve the problem. To improve the quality of the solution a local exchange procedure is also implemented. Sensitivity analysis is conducted on the randomly generated datasets for obtaining the best possible combination of GA parameters. Gao et al. (2009) developed a 0-1 integer programming problem for solving RALB-2 problem. A hybrid genetic algorithm (hGA) is developed to find efficient solutions for the problem. The proposed genetic algorithm uses partial representation technique, which expresses only part of the decision information about a candidate solution in the chromosome. The coding space contains only partial candidate solutions including the optimal one. New crossover and mutation operators are developed to adapt to the nature of the problem. To improve the search ability, local search procedures are also

implemented by them. Yoosefelahi et al. (2012) presented a multi objective model for RALB to minimize the cycle time, robot setup costs and robot costs. A new mixed-integer linear programming model is also developed to solve the problem. Since the problem is NP-hard, three versions of multi-objective evolution strategies (MOES) are employed. It is concluded that hybrid MOES is efficient based on the simulation results obtained. Daoud et al. (2014) proposed several evolutionary algorithms and a discrete event simulation model to solve robotic assembly line balancing problem and an automated packaging line dedicated for dairy food products is the case study considered for evaluating the proposed model.

To provide an overview of this section, Table 2.1 summarizes the literature on RALB Studies are categorized according to the objectives and optimization techniques.

Table 2.1 Summary of research on RALB problems

Model	Objective	Procedure	Reference
Assignment of tasks to non-identical workstations without violating precedence relations.	Minimize the cost of workstations	Dynamic Programming	Nicosia et al. (2002)
RALB-1 problem	Minimize the number of workstations	-	Rubinovitz and Bukchin (1991)
RALB-1 problem	Minimize the number of workstations	Branch & Bound	Rubinovitz et al. (1993)
RALB problem	Minimize the equipment cost	An exact and heuristic branch & bound	Bukchin and Tzur (2000)
RALB-2 problem	Minimize the cycle time	Two versions of Genetic Algorithm	Levitin et al.(2006)
RALB-2 problem	Minimize the cycle time	Hybrid Genetic Algorithm	Gao et al.(2009)
RALB-2 problem	Minimize the cycle time, robot cost, setup costs	Three versions of multi-objective evolution strategies	Yoosefelahi et al.(2012)
RALB -E problem	Maximize line efficiency	Three evolutionary algorithms & Local Search	Daoud et al.(2014)

2.6 State of the Art

From the literature survey done, the key problems addressed under the assembly line balancing problems are as follows:

i. *Optimizing different objectives for a straight assembly line.*

   Different optimization techniques are used to solve straight line assembly line balancing problems with an objective of minimizing cycle time, minimizing the number of workstation, minimizing assembly line cost and maximize the line efficiency. Different optimization techniques used are branch and bound, genetic algorithm, simulated annealing, particle swarm optimization, differential evolution and etc.

ii. *Optimizing different objectives for a U-shaped assembly line.*

   Objective of minimizing cycle time, minimizing number of workstations and maximizing line efficiency in a U-shaped assembly line are done using different optimization techniques. Different metaheuristics are proposed to solve this problem.

iii. *Straight line robotic assembly line balancing problem with different objectives.*

   Robotic assembly line balancing in a straight assembly line with objective of minimizing cycle time, minimizing the number of workstations and maximizing line efficiency are done using few optimization techniques. Metaheuristics like genetic algorithms and its hybrid version is proposed to solve the problem.

In summary, this chapter reports the literature related to assembly line balancing. Literature survey reveals the different types of assembly lines in detail. This chapter provides detailed information of the need of balancing an assembly line in an industrial sector and the importance of this type of research in academics. Different problems classified under the problems are discussed. This survey provides a detailed review of the different solution procedures applied to solve the ALB problems. As discussed in Section 2.5.4 researchers have proposed models and solution procedures for solving robotic assembly line balancing (RALB) problems. The work on RALB is very limited and hence this thesis seeks to develop efficient algorithms to solve problems in RALB with different objectives.

CHAPTER 3

Problem Definition

In this chapter, details of the research problem considered for the study are presented through motivation of research, problem statement, and research objectives.

3.1 Motivation of the research

Due to the increased demand for productivity, quality, cost reduction and optimal utilization of the available resources researchers have motivated to do continuous research in modeling and evaluation of manufacturing systems. Assembly line is one of the major components in manufacturing sector. Availability of different types of robots to perform the assembly tasks, led to the development of automated assembly line. Research interest in robotic/automated/flexible assembly is the main motivation of this research. The research interest surrounding Flexible Manufacturing System, Flexible Assembly System, Computer Integrated Manufacturing, and application of metaheuristic algorithms to solve the problems from these systems has attracted many researchers to work in the area of assembly line balancing problems

The topic has become relevant in the present day, since the international manufacturing strategy and operations are inclined towards the research of automated assembly systems and considers it strategically important at the professional forefront. There is a growing demand and importance for automated assembly systems, which require a robust integrative technological perspective and pragmatic approach for long term investment.

In the recent years, different metaheuristics have been proposed to solve assembly line balancing problems. Different types of bio-inspired algorithms are genetic algorithms, simulated annealing, differential evolution, ant and bee algorithms, bat algorithm, particle swarm optimization, harmony search, firefly algorithm, cuckoo search and others. This research focuses on using PSO, DE and Cuckoo search for solving RALB problems. There has been very minimal research done on robotic assembly line balancing problem with different objectives. Hence the motivation of this

study is also focused towards the development of different variations of RALB problems.

3.2 Problem statement

Even though the benefits of using robotic/ automated assembly line are substantial, it can be inferred from the literature that only few papers describe the design and optimization of robotic assembly line balancing (RALB) problems. To the best of the author's knowledge the following important aspects has not been reported till date. Important aspects of RALB which did not find place in the existing literature are as follows:

i. Modeling and optimization of U-shaped robotic assembly line balancing problems with different objectives.
ii. Modeling and optimization of energy based robotic assembly line balancing problems for a straight and U-shaped robotic assembly line.
iii. Modeling and optimization of cost based robotic assembly line balancing problems for a straight and U-shaped robotic assembly line.

Henceforth, the research is to address and understand the above mentioned shortcomings and to develop efficient algorithms for solving robotic assembly line balancing problems to obtain better quality solution.

3.3 Research objectives

Based on the findings from the literature survey, the following have been set as the objectives of this research:

1. To develop efficient metaheuristic algorithms to find best solution for the straight robotic assembly line balancing problems with an objective of minimizing the cycle time when number of workstations are fixed and compare the performance of results obtained through these algorithms with results published in literature.
2. To develop an efficient algorithm for a U-Shaped Robotic Assembly Line with the objective of minimizing the cycle time.
3. To develop an efficient metaheuristic algorithm for solving energy based robotic assembly line problem for both straight and U-shaped robotic assembly line.

4. To develop an efficient algorithm for solving cost based robotic assembly line balancing problem in both straight and U-shaped robotic assembly line.
5. To develop efficient metaheuristics algorithm to solve a robotic assembly line problem with an objective of maximizing the line efficiency.

The scope of the current study is limited to the development of efficient metaheuristic algorithms for solving robotic assembly line balancing problems.

3.4 Research approach

Following steps are adopted for developing efficient models for solving robotic assembly line balancing problems.

1. Design and development of metaheuristic solution procedure to solve RALB problem using different metaheuristic algorithms.
2. Validate the developed model by evaluating the benchmark literature datasets and datasets which are generated based on the literature.
3. Investigate the performance of the proposed metaheuristic by comparing the reported results using other solution procedures.
4. Performing parametric study for different metaheuristic algorithms to find out the best combination of parameters for solving the problem effectively
5. Drawing conclusions and discussing the directions for future work.

The motivation of research, problem statement, research objectives and the proposed research approach are presented in detail in this chapter. Next chapter presents the assumptions and mathematical model for RALB problems in detail.

CHAPTER 4

Mathematical Models for RALB Problems

The objectives of the research are presented in the previous chapter. This chapter provides the details of the mathematical models for robotic assembly line balancing problems with different objectives. Assumptions and notations considered for different problems are also listed in this chapter.

4.1 Straight RALB problem - minimizing cycle time

Robotic assembly lines are used by manufacturers for producing high volume product and to produce products with high quality. Workstations are connected together with a material handling system for an assembly line. An assembly line helps at assembling components into a final product. At each workstation a set of tasks are to be performed to assembly a product. The precedence constraints need to be specified and it specifies the order in which the tasks are to be executed. The assembly line system must be configured for the assembly of the product where tasks are assigned to the workstations and best available robot is allocated to the workstation with an objective of minimizing the cycle time of the assembly line. Mathematical model is developed based on the objective of minimizing cycle time when the number of workstations is fixed for a straight robotic assembly line.

4.1.1 Assumptions and Mathematical Model

The following assumptions are considered while developing the mathematical model. The assumptions considered in this model are similar to those followed by Levitin et al. (2006) and Gao et al. (2009).

- A unique model of single product is assembled on a straight robotic assembly line.
- Tasks cannot be subdivided and tasks can be processed only if the task sequence meets the precedence requirements.
- Tasks cannot be shared among other workstations.
- Time taken for performing a task depends on the type of the robot allocated.

- Robot type to be assigned to a workstation is selected among other types of robot based on the time taken by the robot to perform the tasks allocated within minimum time.
- At a time only one robot could be assigned to a workstation.
- Number of workstations is equal to the number of available robots.
- Any task can be performed at any workstation and by any robot if the precedence relation is not violated. Task is carried out by a robot at a workstation where it is assigned.
- There is no limitations in the availability of the robots (i.e. number of robots of same capability is unrestricted).
- Cost of purchasing the robot is not considered.
- In a single model assembly line, the material handling, loading and unloading time, as well as set-up and tool changing time are negligible, because the tooling changes are minimized in a robotic assembly line. This assumption is realistic on a single-model assembly line. If tool change or other type of set-up activity is necessary, it can be included in the task time.

Levitin et al. (2006) mentioned two main objectives in their paper: Optimal balance of the assembly line and allocation of the best fit robot to each workstation. Achieving these two objectives are only possible when the assumption that robot type is selected based on the time taken by the robot to perform the tasks allocated with minimum time is considered. Since this consideration helps to reduce the cycle time of the assembly line and assign a best fit robot to workstations, it is followed in this model. The model presented below is the modification of the one presented by Gao et al. (2009) considering the assumption presented above. The zero-one integer programming (IP) model for the problem of minimizing cycle time for a straight robotic assembly line is presented below.

- *Decision Variables*

$$x_{is} = \begin{cases} 1 \text{ if task } i \text{ is assigned to workstation } s \\ 0, \text{ otherwise} \end{cases}$$

$$y_{sh} = \begin{cases} 1 \text{ if robot } h \text{ is allocated to workstation } s \\ 0, \text{ otherwise} \end{cases}$$

- *Zero-One Integer formulation:*

$$\min c = \max_{1 \le s \le N_w} \left\{ \sum_{i=1}^{N_a} \sum_{i=1}^{N_w} t_{ih} . x_{is} . y_{sh} \right. \quad \textbf{(4.1)}$$

$$s.t. \sum_{s=1}^{N_w} s.x_{is} - \sum_{s=1}^{N_w} s.x_{js} \le ,0 \quad \forall i \in pre(j); j \quad \textbf{(4.2)}$$

$$\sum_{s=1}^{N_w} x_{is} = 1 \qquad \forall i \quad \textbf{(4.3)}$$

$$\sum_{s=1}^{N_w} y_{sh} = 1 \qquad \forall s \quad \textbf{(4.4)}$$

$$x_{is} \in \{0,1\} \quad \forall s,i \quad \textbf{(4.5)}$$

$$y_{sh} \in \{0,1\} \quad \forall h,s \quad \textbf{(4.6)}$$

The objective Equation 4.1 is to minimize the cycle time of the robotic assembly line. Equation 4.2 defines the precedence relationship among the tasks. It ensures that for a pair of tasks with precedence relation, the precedent cannot be assigned to a workstation after the one to which its successor is assigned. Equation 4.3 ensures that each task has to be assigned to one workstation and Equation 4.4 ensures that each workstation is equipped with one robot. It is notable that objective is non-linear. Hence, it is hard for traditional exact optimization techniques to solve the problem.

4.2 U-shaped RALB problem - minimizing cycle time

Assembly tasks of different types are performed at each work station. These tasks are to be completed to produce a final product. Precedence constraints are specified and it determines the order in which tasks should be executed. Tasks are to be assigned to the work station and the best robot needs to be allotted to the work station to perform the tasks. The assembly of the tasks is to be performed in a U-shaped robotic assembly line with a main objective of minimizing the cycle time.

4.2.1 Assumptions and Mathematical Model

The assumptions used for the mathematical model for U-shaped robotic assembly line are same as the one mentioned in Section 4.1.1 and these assumptions are based on the assumptions used by Levitin et al. (2006) and Gao et al. (2009) for RALB problems. Based on the definition proposed by Gutjahr and Nemhauser (1964) for a straight line assembly line problem, Miltenburg and Wijngaard (1994) presented a definition for the simple U-shaped assembly line problem. Gao et al. (2009) presented a formulation for type-II RALB. The formulation presented in this thesis is based on these definitions. The model presented is for U-shaped robotic assembly line balancing problem with an objective of minimizing cycle time when number of workstations is fixed.

Zero-one integer programming (IP) model for this problem is formulated as follows:

The problem is, for a given set of tasks $F = \{g \mid g = 1, 2, .. ., n\}$, a set of precedence constraints $P = \{ (i, j) \mid$ task i must be completed before task $j \}$, a set of task times $T = \{ t(g) \mid g = 1, 2, ..., n \}$, and a cycle time C, find a collection of subsets of F, $(L_1, L_2, . . . , L_N)$ where $L_a = \{g \mid$ task g is done at workstation $a \}$ and the workstations and tasks are arranged in a U-shape.

- *Decision Variables*

$$x_{is} = \begin{cases} 1 \text{ if task } i \text{ is assigned to workstation } s \\ 0, \text{ otherwise} \end{cases}$$

$$y_{sh} = \begin{cases} 1 \text{ if robot } h \text{ is allocated to workstation } s \\ 0, \text{ otherwise} \end{cases}$$

- *Zero-One Integer formulation:*

$$\min c = \max_{1 \le s \le N_w} \left\{ \sum_{i=1}^{N_a} \sum_{i=1}^{N_w} t_{ih} . x_{is} . y_{sh} \right. \tag{4.7}$$

For each task j:

$$\begin{aligned} &\text{if } (i, j) \in P, i \in L_a, j \in L_b, \text{ then } a \le b, \text{ for all } i; \text{or} \\ &\text{if } (j, k) \in P, y \in L_b, k \in L_c, \text{ then } c \le b, \text{ for all } k; \end{aligned} \tag{4.8}$$

$$\sum_{s=1}^{N_w} x_{is} = 1 \qquad \forall i \tag{4.9}$$

$$\sum_{s=1}^{N_w} y_{sh} = 1 \qquad \forall s \tag{4.10}$$

$$x_{is} \in \{0,1\} \quad \forall s,i \tag{4.11}$$

$$y_{sh} \in \{0,1\} \quad \forall h,s \tag{4.12}$$

The objective of Equation 4.7 is to minimize the cycle time of the robotic assembly line. Equation 4.8 ensures the precedence constraints are not violated on the U-shaped assembly line. Equation 4.9 ensures that each task has to be assigned to one workstation and Equation 4.10 ensures that each workstation is equipped with one robot. Objective function is non-linear. Hence, it is hard for traditional exact optimization techniques to solve the problem.

4.3 RALB problem - minimizing energy consumption

In an assembly line, different assembly tasks are to be performed by each workstation to assemble and produce a given product, while precedence constraints of the tasks are specified. A set of workstations and robots are considered in the assembly line. In a balanced assembly line, tasks needs to be assigned to the workstations and best robot needs to be allotted to the station to perform the assembly tasks with minimum energy consumption. The mathematical model presented here is similar to the one presented in previous sections except the objective function. The objective in this model is to minimize the energy consumption in straight and U-shaped robotic assembly line. Other than the assumptions presented in Section 4.1.1 and 4.2.1 few new assumptions are incorporated for developing this model.

4.3.1 Assumptions and Mathematical Model

This section presents the new set of assumptions used for the mathematical model development.

Assumptions considered are:

- Robots power consumptions are assumed. Using the power of each robot, energy consumption is calculated.

- The planning horizon is not included in the model. The proposed algorithm and the models are tested using the benchmark problems available in the literature. Hence, the maintenance operations are not considered in this study.

According to the assumptions considered, a zero-one integer programming (IP) model for this problem is formulated as follows:

- *Decision Variables*

$$x_{is} = \begin{cases} 1 \text{ if task } i \text{ is assigned to workstation } s \\ 0, \text{ otherwise} \end{cases}$$

$$y_{sh} = \begin{cases} 1 \text{ if robot } h \text{ is allocated to workstation } s \\ 0, \text{ otherwise} \end{cases}$$

- *Zero-One Integer formulation:*

$$\min E = \sum_{i=1}^{Nw} \left\{ \sum_{i=1}^{N_a} \sum_{i=1}^{N_w} e_{ih}\, x_{is} y_{sh} \right. \quad \textbf{(4.13)}$$

$$s.t. \sum_{s=1}^{N_w} s.x_{is} - \sum_{s=1}^{N_w} s.x_{js} \leq ,0 \quad \forall i \in pre(j); j \quad \textbf{(4.14)}$$

$$\sum_{s=1}^{N_w} x_{is} = 1 \qquad \forall i \quad \textbf{(4.15)}$$

$$\sum_{s=1}^{N_w} y_{sh} = 1 \qquad \forall s \quad \textbf{(4.16)}$$

$$x_{is} \in \{0,1\} \quad \forall s,i \quad \textbf{(4.17)}$$

$$y_{sh} \in \{0,1\} \quad \forall h,s \quad \textbf{(4.18)}$$

The objective of the energy based model (Equation 4.13) is to minimize the total energy consumption. Equation 4.14 defines the precedence relationship among the tasks. It ensures that for a pair of tasks with precedence relation, the precedent cannot be assigned to a workstation after the one to which its successor is assigned. Equation 4.15 ensures that each task has to be assigned to one workstation and Equation 4.16 ensures that each workstation is equipped with one robot. Objective is non-linear. Hence, it is hard for traditional exact optimization techniques to solve the problem.

The model presented is for straight robotic assembly line, in case of U-shaped robotic assembly line precedence relationship equation changes. Hence Equation 4.14 is replaced by Equation 4.19.

For each task *j*:

$$\begin{aligned} &\text{if } (i, j) \in P, i \in L_a, j \in L_b, \text{then } a \le b, \text{ for all } i; \text{or} \\ &\text{if } (j, k) \in P, y \in L_b, k \in L_c, \text{then } c \le b, \text{ for all } k; \end{aligned} \qquad \textbf{(4.19)}$$

4.4 RALB problem - minimizing assembly line cost

A robotic assembly line consists of several workstations, each of them being responsible for performing a specific set of tasks done by a robot. The cost of performing the tasks is different from robot to robot, since the robots are manufactured from different vendors with different capabilities and specifications. Selection of most suitable robot among many alternatives to perform a set of tasks in a particular workstation with a specific objective is a typical optimization problem. RALB problem considered here is to find the optimal set of tasks allotted to each work stations such that precedence constraints between tasks or other constraints are met and selection of suitable robot type to execute the tasks. The objective considered here is minimization of overall assembly line cost.

4.4.1 Assumptions and Mathematical Model

The following assumptions are considered in the model formulation of proposed robotic assembly line balancing problem.

- Robot initial costs are assumed. It includes installation, maintenance and service cost for the entire service life. The service life is limited to 5 years.
- All robots are working 20hrs a day and 300 days in a year.
- Equivalent uniform annual costs of all robots are calculated with annual fixed interest rate @10%.

A zero-one integer programming (IP) model for this problem is formulated as follows:

- *Decision Variables*

$$x_{is} = \begin{cases} 1 \text{ if task } i \text{ is assigned to workstation } s \\ 0, \text{ otherwise} \end{cases}$$

$$y_{sh} = \begin{cases} 1 \text{ if robot } h \text{ is allocated to workstation } s \\ 0, \text{ otherwise} \end{cases}$$

- *Zero-One Integer formulation:*

$$\min Cost = \sum_{i=1}^{N_w} \left\{ \sum_{i=1}^{N_a} \sum_{i=1}^{N_w} c_{ih}\, x_{is} y_{sh} \right. \quad \textbf{(4.20)}$$

$$s.t. \sum_{s=1}^{N_w} s.x_{is} - \sum_{s=1}^{N_w} s.x_{js} \leq, 0 \quad \forall i \in pre(j); j \quad \textbf{(4.21)}$$

$$\sum_{s=1}^{N_w} x_{is} = 1 \qquad \forall i \quad \textbf{(4.22)}$$

$$\sum_{s=1}^{N_w} y_{sh} = 1 \qquad \forall s \quad \textbf{(4.23)}$$

$$x_{is} \in \{0,1\} \quad \forall s,i \quad \textbf{(4.24)}$$

$$y_{sh} \in \{0,1\} \quad \forall h,s \quad \textbf{(4.25)}$$

The objective of the energy based model (Equation 4.20) is to minimize the total assembly line cost. Equation 4.21 defines the precedence relationship among the tasks. It ensures that for a pair of tasks with precedence relation, the precedent cannot be assigned to a workstation after the one to which its successor is assigned. Equation 4.22 ensures that each task has to be assigned to one workstation and Equation 4.23 ensures that each workstation is equipped with one robot. It is notable that objective function is non-linear. Hence, it is hard for traditional exact optimization techniques to solve the problem.

The model presented is for straight robotic assembly line, in case of U-shaped robotic assembly line precedence relationship equation changes. Hence Equation 4.21 is replaced by Equation 4.26.

For each task *j*:

$$\begin{aligned} &\text{if } (i,j) \in P, i \in L_a, j \in L_b, \text{then } a \leq b, \text{ for all } i; \text{or} \\ &\text{if } (j,k) \in P, y \in L_b, k \in L_c, \text{then } c \leq b, \text{ for all } k; \end{aligned} \quad \textbf{(4.26)}$$

4.5 RALB problem - maximizing line efficiency

Different assembly tasks are performed by each station to produce a final product in an assembly line. Assembly lines are of two layouts (straight and U-shaped layout). In a balanced robotic assembly line, tasks are assigned to workstation and the best robot is allotted to the workstation to perform the task allotted. Major objective of this problem is to achieve maximum line efficiency for the assembly line. In case of straight robotic assembly line, the set of possible assignable tasks are decided by those tasks whose predecessors are already assigned. In case of U-shaped robotic assembly line, the set of assignable tasks is determined by all those tasks whose predecessors or successors have already been assigned.

4.5.1 Assumptions and Mathematical Model

Assumptions considered for development of this model is similar to the assumption used for developing the mathematical model where the objective of minimizing the cycle time is considered.

A zero-one integer programming (IP) model for this problem is formulated as follows:

- *Decision Variables*

$$x_{is} = \begin{cases} 1 \text{ if task } i \text{ is assigned to workstation } s \\ 0, \text{ otherwise} \end{cases}$$

$$y_{sh} = \begin{cases} 1 \text{ if robot } h \text{ is allocated to workstation } s \\ 0, \text{ otherwise} \end{cases}$$

- *Zero-One Integer formulation:*

$$\max Z = LE \qquad \textbf{(4.27)}$$

$$s.t. \sum_{s=1}^{N_w} s.x_{is} - \sum_{s=1}^{N_w} s.x_{js} \leq 0 \quad \forall i \in pre(j); j \qquad \textbf{(4.28)}$$

$$\sum_{s=1}^{N_w} x_{is} = 1 \qquad \forall i \qquad \textbf{(4.29)}$$

$$\sum_{s=1}^{N_w} y_{sh} = 1 \qquad \forall s \qquad \textbf{(4.30)}$$

$$x_{is} \in \{0,1\} \quad \forall s,i \tag{4.31}$$

$$y_{sh} \in \{0,1\} \quad \forall h,s \tag{4.32}$$

The objective of the model is to maximize the line efficiency. Equation 4.28 defines the precedence relationship among the tasks. It ensures that for a pair of tasks with precedence relation, the precedent cannot be assigned to a workstation after the one to which its successor is assigned in case of straight robotic assembly line. Equation 4.29 ensures that each task has to be assigned to one workstation and Equation 4.30 ensures that each workstation is equipped with one robot. Objective function of the model is non-linear and therefore it is hard for traditional exact optimization techniques to solve the problem.

The model presented is for straight robotic assembly line, in case of U-shaped robotic assembly line precedence relationship equation changes. Hence Equation 4.28 is replaced by Equation 4.26. Line efficiency of a given assembly line is the direct indication of the efficiency (Khaw and Ponnambalam, 2009). The line efficiency is calculated as follows.

$$LE = \frac{\sum_{k=1}^{N_w} S_k}{N_w * c} * 100 \tag{4.33}$$

Where S_k is the k^{th} workstation time, N_w is the total number of workstations and c is the cycle time.

4.6 Summary

The assumptions and mathematical models proposed for different robotic assembly line balancing problems with different objective functions are presented in this chapter.

Zero-One Integer formulation is presented for the objective of minimizing the cycle time in a straight robotic assembly line. Assumptions considered to solve the problem are also presented. The chapter presents the mathematical model for U-shaped robotic assembly line with the objective of minimizing the cycle time. The set of

assumptions for this model is same as the one presented for the straight robotic assembly line.

Mathematical model to minimize the energy consumption in a robotic assembly line is presented. The assumptions considered for this model is also presented. The variation needed in the model when the layout of the assembly line considered is U-shaped is also explained.

The mathematical model for the objective of minimizing the assembly line cost is also discussed along with the assumptions considered. The model is presented for the straight and U-shaped layout of robotic assembly line.

Mathematical model for the objective of maximizing line efficiency in a straight and U-shaped robotic assembly line is also presented. The assumptions for this model are same as the assumptions used for the model where the objective is to minimize the cycle time.

CHAPTER 5

Particle Swarm Optimization & Hybrid Particle Swarm Optimization for RALB Problem to Minimize Cycle Time

Metaheuristics have been widely used for solving combinatorial optimization problems. This is mainly due to the increasing computational speed of computers, which helps the use of metaheuristics to solve real world problems. Metaheuristics are often hybridized with local search methods to improve the rate of convergence. Different metaheuristics such as genetic algorithm, simulated annealing and tabu search are proposed in the literature to solve assembly line balancing problems. In this section different metaheuristics like Particle swarm optimization, PSO Variants and hybrid Cuckoo search- PSO are proposed to solve robotic assembly line balancing problems with objective of minimizing the cycle time of the robotic assembly line. The model will be referred to as RALB-2 in this thesis. The solution to the RALB problem includes an attempt for optimal assignment of robots to line stations and a balanced distribution of work between different stations.

In this research metaheuristic algorithms like particle swarm optimization, differential evolution and hybrid algorithms are implemented to solve robotic assembly line balancing problems with an objective of minimizing cycle time. PSO is used due to the easiness in implementation, faster convergence and very less parameters to fine tune (Wu et al., 2011). Hybridized metaheuristics are also implemented to solve RALB problems. Search capability of the algorithm is improved by hybridizing metaheuristics and this helps to improve the quality of the solution.

5.1 Standard Particle Swarm Optimization for straight RALB

In the recent years, extensive study has been done to understand the social psychology of fish schools, birds flock and bug swarm. It is observed that social behavior play a very crucial role in the survival of species and its adaptation to the environment change. Particle swarm optimization (PSO) is a relatively new approach in the modern metaheuristics for optimization. PSO is one of the evolutionary

computation methods. Particle Swarm Optimization (PSO) is a computational method developed by Kennedy and Eberhart (1995). This method is motivated by simulation of social behavior. This algorithm has been widely used to solve optimization problems. Ease of implementation, robustness and very few parameters to adjust makes PSO very attractive among researchers. PSO is a population based search algorithm and is initialized with a random set of population solutions, called particles (swarm). Each particle in PSO is associated with a velocity.

PSO originates from social psychology as a simulation of socio-cognitive processes to model abstract concept of swarm intelligence that has the following several advantages:

- Easy to implement
- Few parameters to fine tune
- Relatively smaller population size
- Relatively small number of function evaluations to converge
- Faster computation

Particles fly through the search space with a specified velocity which is dynamically adjusted based on their historical behaviors. For each iteration *'t'*, each particle *'i'* keeps tracks of its coordinates in the problem space which are associated with the best solution it has achieved so far. This value is called local best (${}^{e}P^{t}_{i}$). Another best value which is tracked is the overall best value, and its location obtained so far by any particle in the population. This location is called global best (G). Based on the concept proposed by Kennedy and Eberhart, in every iteration, there is a change in the velocity of each particle which makes the move towards possibly new ${}^{e}P^{t}_{i}$ and G^{t}. Pseudo code of the standard PSO algorithm is presented in Figure 5.1.

```
for each particle
   Initialize particle
end
do
   for each particle
      Calculate fitness value
      If the fitness value is better than the best fitness value (Local Best) in history
         set current value as the new Local Best
end
   Choose the particle with the best fitness value of all the particles as the Global Best
   for each particle
      Calculate particle velocity according Equation 5.1
      Update particle position according Equation 5.2
   end
while maximum iterations criteria is not attained
```

Figure 5.1 Pseudo code of standard PSO

The generic PSO algorithm for a problem is given below:

Step 1: Initialization of the populations: Generate set of particles with random position or values and initial velocities.

Step 2: For each particle, evaluate the optimization function yielded by the particle.

Step 3: Initialize the particle best. Each particle remembers the best result achieved so far (personal best) and exchanges information with other particles to determine the best particle (global best) among the swarm.

Step 4: Apply velocity and move the particle according to Equation 5.1 and Equation 5.2, respectively:

Velocity update equation:

$$v_i^{t+1} = \underbrace{c_1 v_i^t}_{\text{Momentum Part}} + \underbrace{c_2 \text{ x } [U_1 \text{ x } ({}^e P_i^t - P_i^t)]}_{\text{Cognitive Part}} + \underbrace{c_3 \text{ x } [U_2 \text{ x } (G^t - P_i^t)]}_{\text{Social Part}} \quad \textbf{(5.1)}$$

Particle's moves from their current position to the new position using Equation 5.2 Each particle's position is updated in each generation by adding the velocity vector to the position vector.

Position update equation:

$$P_i^{t+1} = P_i^t + v_i^{t+1} \quad \textbf{(5.2)}$$

Where U_1 and U_2 are known as velocity coefficients (random numbers between 0 and 1), c_1, $c_2$and c_3 are known as acceleration coefficients, v_i^t is the initial velocity, ${}^eP^t_i$ is the Local best, G^t is the global best solution at generation *'t'* and P_i^t is the current particle position. The parameters U_1, U_2 are used to maintain the diversity of the population, and they are uniformly distributed in the range of zero and one and the values of U_1 and U_2 varies for all iterations.

Step 5: Go back to step until the termination criterion is met.

The Equation 5.1 and 5.2 describe the trajectory in which the particles fly. Equation 5.1 describes how the velocity is updated and Equation 5.2 describes the position update of the particle which is flying in the search space. Equation 5.1 consists of three parts. First part is known as momentum part. The velocity cannot be changed abruptly and it is changed from the current velocity. Second part is known as cognitive part which signifies the private thinking of itself learning from its own flying experience. Third part is known as social part which represents the collaboration among particles learning from the group flying experience (Blondin, 2009).

In Equation 5.2, if the sum of three parts on the right side exceeds a constant value specified by the user, then the velocity on that dimension is assigned to be v_{max} that is particles velocity is restricted to a maximum velocity v_{max} which is an important parameter. Large value of v_{max} helps the particles to fly past the good solution areas. Small v_{max} might lead particles to fall in the local minima, which makes them unable to fly into better solution area. From the literature, it could be observed that v_{max} is usually set as a constant value. But researchers have developed dynamically changing v_{max} for improving the performance of PSO. The velocity calculation applied in this research is adopted from the PSO algorithm developed by Clerc (2004).

Various operations performed for computing particle velocity and updating particle positions (Rameshkumar et al., 2005) are explained below:

Subtraction (position – position) operator: Let us assume to positions x_1 and x_2 representing two different task sequences. The difference of x_2 - x_1 is a velocity v. In the Equation 5.1, for example subtracting two positions i.e. $({}^ep_i^t - P_i^t)$ results in a velocity which is a set of transpositions.

Addition (position + velocity) operator: Let us assume x to be the position and v to be the velocity of the particle. New position x_1 is calculated by applying the first transposition of v to x, i.e, $x_1 = x + v$ then the second one to the result etc.

Addition (velocity + velocity) operator: Let us assume two velocities v_1 and v_2. In order to calculate $v_1 + v_2$, the list of transpositions which contains first the 'ones' of v_1, followed by the 'ones' of v_2 is considered.

Multiplication (Coefficient × velocity) operator: Let c be the learning coefficient and v be the velocity. $c \times v$ results in a new velocity.

Let us assume the following data for a numerical illustration of velocity and position update (without reference to iteration) in a RALB problem with 11 tasks:

Local Best ${}^{e}P^{t}_{i}$ *:* (1,2,6,3,4,5,7,8,10,9,11), *Global Best G:*(1,2,3,4,5,6,7,8,9,10,11)

Particle P_i^t *:* (1,2,3,6,5,4,7,8,10,9,11) and *Initial velocity* v_i^t*:* (2, 3) (4, 5).

Initial value of velocity is randomly generated with length of the velocity pair restricted to 2. Similarly, velocity index is computed for other particles. These sequences represent 11 task problem arranged as per the precedence order for RALB problem. The number of velocity pair for the problem considered is 2. Parameters chosen for the example: c_1=1, c_2=1 and c_3=2 are the acceleration coefficients and U_1, U_2, U_3 ranges between 0 and 1. Let U_1=0.8, U_2=0.3. Using Equation 5.1 the velocity of the particle is calculated. If the coefficient value is 0.8, then 80 percent of the velocity components are randomly selected and are applied to generate velocity of the particle and similarly position of the particle is also updated.

The velocity for the particle is updated using Equation 5.1.

v_i^{t+1} = (2,3)(4,5)+0.8 x [(1,2,6,3,4,5,7,8,10,9,11)–(1,2,3,6,5,4,7,8,10,9,11)]+0.6 x [(1,2,3,4,5,6,7,8,9,10,11)-(1,2 ,3,6,5,4,7,8,10,9,11)]

= (2,3)(4,5)+0.8 x (2,3)(4,5) +0.6 x (3,5)(8,9) = (2,3)(4,5)(8,9)

Position of the particle '*i'* is updated using Equation 5.2. Particles move from their current position to the new position.

P_i^{t+1} = (1,2,3,6,5,4,7,8,10,9,11)+(2,3) (4,5)(8,9) = (1,3,2,5,6,4,7,10, 8,9,11)

In this chapter, PSO, PSO variants and hybridized Cuckoo Search-PSO is proposed to solve RALB problems. Results obtained from these algorithms are compared with benchmark results obtained for RALB problem with an objective of minimizing the cycle time.

5.1.1 Implementation of PSO

The implementation details of PSO on RALB-2 problem with an objective of minimizing cycle time are explained in this section.

a) Population Initialization

Metaheuristic algorithms generally start with a randomly generated search space (Huang et al., 2014) which evolves iteratively to find nearer to optimal solutions. Instead of starting the search process from a set of random solutions, a set of priority rules reported in the literature are used to reach better solution at a faster rate.

Table 5.1 Initial population generated using the heuristic rules

Methods	**Particle Generated**										
Maximum Rank Positional Weight	1	2	6	3	4	5	7	8	10	9	11
Minimum Inverse Positional Weight	1	5	4	3	2	7	9	6	8	10	11
Minimum Total Number Of Predecessors Tasks	1	2	3	4	5	6	8	10	7	9	11
Maximum Total Number of Follower Tasks	1	2	3	4	5	6	7	8	9	10	11
Maximum Task Time	1	5	2	6	3	4	7	8	10	9	11
Minimum Task Time	1	4	3	2	5	7	9	6	8	10	11

Table 5.2 Performance time for 11 tasks by 4 robots/workers

Tasks	**Robot 1**	**Robot 2**	**Robot 3**	**Robot 4**	**Average Time (τ)**
1	81	37	51	49	54.5
2	109	101	90	42	85.5
3	65	80	38	52	58.75
4	51	41	91	40	55.75
5	92	36	33	25	46.5
6	77	65	83	71	74
7	51	51	40	49	47.75
8	50	42	34	44	42.5
9	43	76	41	33	48.25
10	45	46	41	77	52.25
11	76	38	83	87	71

Six particles are generated using the six heuristic rules (Ponnambalam et al., 2000): maximum rank positional weight, minimum inverse positional weight, minimum total number of predecessors tasks, maximum total number of follower tasks, maximum and minimum task time. Remaining swarm particles are randomly generated. There are

twenty five particles in the initial swam. Table 5.1 shows set of particles formed using those methods and remaining particles are generated randomly which are satisfying the precedence condition. The precedence graph shown in Figure 5.2 and processing time shown in Table 5.2 are used to generate the information provided in Table 5.1.

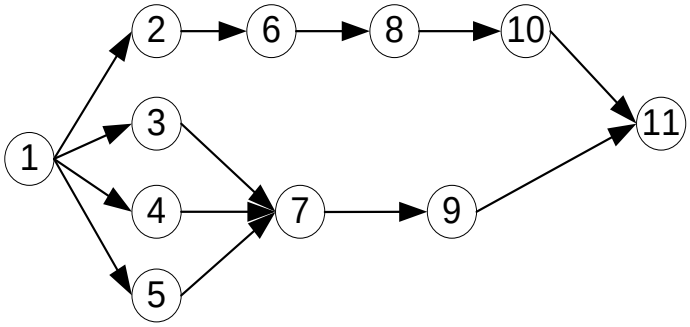

Figure 5.2 Precedence Graph of 11 task problem

It is to be noted that the particle structure is a string, consists of tasks to be performed in RALB problem satisfying the precedence constraints. After the swarm is initialized, each particle is assigned with random velocity and length of the velocity of each particle is generated randomly which is explained in the next section. The six heuristic rules used are explained with an example for better understanding.

1. Maximum Rank Positional Weight

Each task is prioritized based on the cumulative assembly time associated with itself and its successors. The steps for forming the sequence using Maximum Rank Positional Weight are as below:

1. Let *S (i)* ➔ Set of successors of tasks *i*.
2. Tasks ordered such that $i < r$ implies $i \text{ not} \in S(r)$.
3. Task *r* is then a member of *S (i)* only if there is an immediate successor relationship from *i* to *r*.
4. Each task has its own task time using which the Positional Weight of a particular task i is calculated.

$$Pw_i = t + \sum_{r \in S(i)} t_r \qquad \textbf{(5.3)}$$

5. Based on the positional weight the tasks are ranked in descending order. Sequence formed based on the rank is: 1 2 6 3 4 5 7 810 9 11. Table 5.3 shows the rank calculated based on positional weight.

Table 5.3 Maximum Rank Positional Weight

Tasks	Average	Immediate Successors	Pw_i	Rank
1	54.5	2,3,4,5	344.75	1
2	85.5	6	290.25	2
3	58.75	7	190.75	4
4	55.75	7	187.75	5
5	46.5	7	178.5	6
6	74.0	8	204.75	3
7	47.75	9	132	7
8	42.5	10	130.75	8
9	48.25	11	84.25	10
10	52.25	11	88.25	9
11	36.0	0	36	11

2. Minimum Inverse Positional Weight

Positional Weight of each task is calculated based on its immediate predecessor's cumulative task time.

1. Let *P (i)* ➔ Set of predecessors of tasks *i*.
2. Tasks ordered such that $i < r$ implies $i \in P(r)$.
3. Task *i* is then a member of *P(r)* only if there is an immediate predecessor relationship from *r* to *i*.
4. Each task has its own task time using which the Positional Weight of a particular task *i* is calculated.

$$Pw_i = t + \sum_{r \in P(i)} t_r \qquad \textbf{(5.4)}$$

6. Based on the positional weight the tasks are ranked in ascending order. Sequence formed is 1 5 4 3 2 7 9 6 8 10 11. Table 5.4 shows the rank calculated based on positional weight.

Table 5.4 Minimum Inverse Positional Weight

Tasks	Average Time	Immediate Predecessors	Pw_i	Rank
1	54.5	-	54.5	1
2	85.5	1	140	5
3	58.75	1	113.25	4
4	55.75	1	110.25	3
5	46.5	1	101	2
6	74	2	214	8
7	47.75	3,4,5	161	6
8	42.5	6	256.5	9
9	48.25	7	197	7
10	52.25	8	308.75	10
11	36	10,9	344.75	11

3. Minimum Total Number of Predecessor tasks

Steps involved in this method

1. Find the total number of predecessors of each task.
2. Sort the tasks based on the total number of predecessors in ascending order
3. After sorting the final sequence is formed. Sequence formed is 1 2 3 4 5 6 8 10 7 9 11. Table 5.5 presents the details used for this heuristic.

Table 5.5 Total Number of Predecessor tasks

Tasks	Total Number of Predecessors	Total Number of Predecessors after sorting
1	0	0
2	1	1
3	1	1
4	1	1
5	1	1
6	2	2
7	4	3
8	3	4
9	7	4
10	4	7
11	10	10

4. Maximum Total Number of Follower Tasks

1. Steps involved in this method
2. Find the total number of successors of each task.
3. Sort the tasks based on the total number of successors in descending order
4. After sorting the final sequence is formed. So sequence formed is 1 2 3 4 5 6 7 8 9 10 11. Table 5.6 shows the details used to form a sequence in this heuristic.

Table 5.6 Maximum Total Number of Follower Tasks

Tasks	Total Number of Successors	Total Number of Successors after sorting
1	10	10
2	4	4
3	3	3
4	3	3
5	3	3
6	3	3
7	2	2
8	2	2
9	1	1
10	1	1
11	0	0

5.Maximum Task Time

The average task time is used for the formation of the sequence. Task is sorted in descending order with respective to their average task time. And final sequence is formed based on the precedence condition. Based on the precedence the sequence formed is 1 5 2 6 3 4 7 8 10 9 11. Table 5.7 presents the details of the average task time.

Table 5.7 Maximum Total Number of Follower Tasks

Tasks	Average Task time (Sorted)
2	85.5
6	74
3	58.75
4	55.75
1	54.5
10	52.25
9	48.25
7	47.75
5	46.5
8	42.5
11	36

6.Minimum Task Time

The average task time is used for the formation of the sequence. Task is sorted in ascending order with respective to their average task time. And final sequence is formed based on the precedence condition. So the sequence formed meeting the precedence constraints is 1 4 3 2 5 7 9 6 8 10 11. Table 5.8 presents the details of average task time.

Table 5.8 Maximum Total Number of Follower Tasks

Tasks	Average Task time (Sorted)
11	36
8	42.5
5	46.5
7	47.75
9	48.25
10	52.25
1	54.5
4	55.75
3	58.75
6	74
2	85.5

Remaining particles are formed by forward, backward and double swapping and the precedence constraints are met.

b) Initial Velocity

Initial velocities for the particles are randomly generated and they represent the number of pairs of transpositions. Table 5.9 shows the maximum number of velocity pairs used. From second iteration onwards velocity update Equation 5.1 is used. Based on the problem size the numbers of velocity pairs are selected. From Table 5.9, different task ranges are presented for which the maximum numbers of velocity pairs are presented. The number of velocity pairs is selected for different problems depending on the tasks size. The number of velocity pairs is kept same throughout all generations. For example, if the task size within the range of 0-20, the number of velocity pair to be selected should be 4. If the numbers of velocity pairs are more than the required number of pairs in subsequent generations, the excess pairs are excluded.

Table 5.9 Maximum number of Velocity Pairs

Task Range	Maximum Velocity Pairs
0-20	4
20-40	8
40-60	10
60-80	25
80-100	30
100-120	40
120-140	50
140-200	65
200-300	75

Two procedures are developed for assignment of tasks and robots to different workstations: recursive procedure and a successive assignment procedure. These procedures are used for finding out the cycle time of the assembly line. This is the objective function which is evaluated using PSO. Recursive and consecutive procedures developed for the RALB problem in a straight robotic assembly line, are discussed and illustrated in detail in the following sections

c) Solution Representation and Fitness Evaluation Cycle time minimization for straight RALB

Each sequence represents a solution which corresponds to a particle in PSO. A sample sequence is shown in Figure 5.3(a) is presented for 11 task RALB problem. Each integer in a sequence represents the task in an assembly line which needs to be performed by a robot in a particular work station. The number of tasks to be assigned to each work station is based on the cycle time. The tasks and robot allocation after

decoding the sequence are also shown in Figure 5.3(b). Each integer in the sequence shown in Figure 5.3(a) is ordered according to their technological precedence constraints. Assigning the tasks to workstation and the robot assignment for the workstation is done using two heuristic procedures: recursive assignment and consecutive assignment procedure.

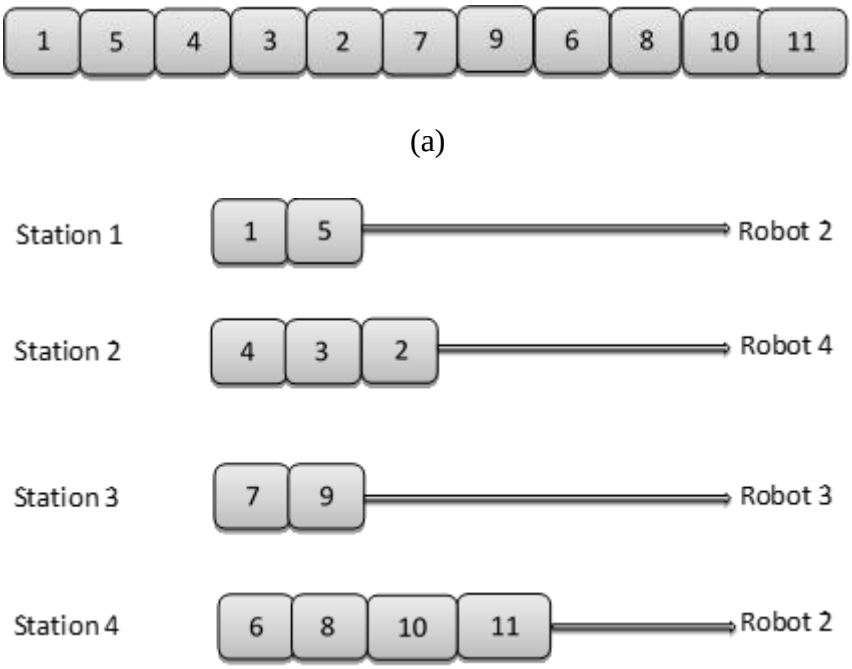

Figure 5.3 a) Sample Task Sequence b) Tasks assigned after decoding the sequence

Cycle time is used as the fitness value to be evaluated. Tasks and robots are assigned to workstation using these heuristic procedures proposed by (Levitin et al., 2006).

Recursive Allocation Method (Cycle Time Calculation)

The method helps in assigning tasks to workstations in such a way the precedence relationship is not violated. The data required for this method are: performance time, number of robots and precedence relationship. Based on the performance time data, using the heuristics particles are generated which is used for the allocation procedure. Average performance time of each task is required for executing this method. Recursive heuristic method is used for finding the cycle time of the assembly line (Levitin et al., 2006).

The recursive procedure divides the sequence '*sq'* into $M = N_w$ parts, and tries to achieve the maximal equality of total execution times for all stations. *'sq'* is the feasible sequence generated based on the heuristics.

The average performance time for each task '*i*' is evaluated:

$$\tau_i = \sum_{h=1}^{N_r} t_{ih}\delta_{h,i} \Big/ \sum_{h=1}^{N_r} \delta_{h,i} \qquad \textbf{(5.5)}$$

where $\delta_{h,i} = 0$ if $t_{ih} = \infty$ and $\delta_{h,i} = 1$ otherwise.

The procedure divides the sequence into two parts (based on the set of elements from the left position *pl=1* to the right position $pr=N_a$) in such a way that it satisfies the A/B ratio, where A= [M/2] and B =M -A

To find this position *i* $(pl \le i \le pr)$ such that Time Ratio value (*TR*).

$$TR = \sum_{j=pl}^{i} \tau_{sq(j)} \Big/ \sum_{j=i+1}^{pr} \tau_{s(j)} \qquad \textbf{(5.6)}$$

The value of time ratio *(TR)* should be as close as possible to the ratio *A/B*.

Using Equation 5.6, value of *TR* is calculated when the initial sequence is divided into two parts where, *pl=1;pr=i* and $pl=i+1;pr=N_a$. Resulting parts are further divided into *M=A* and *M = B* parts respectively using the same procedure recursively until *M* reaches 1. At the end of the recursion, the sequence is divided based on the above conditions and the workstations are fixed. Based on the task allocation, procedure allocated robots which can perform the allocated tasks in minimum performance time.

An example of the procedure is presented in Figure 5.4 and Figure 5.5 based on the performance time data available in Table 5.2 and precedence graph in Figure 5.2. Figure 5.4 presents the recursive allocation procedure for the sequence of tasks ***1-3-2-4-5-6-7-9-8-10-11***; this sequence meets the precedence relationship. From the Figure 5.4 task to be allocated to each workstation is identified and in Figure 5.5 the shaded boxes represents the robot which needs to be selected for performing the tasks allocated to the workstation. Robots are selected based on the minimum robot performance time. The values below the boxes in Figure 5.5 shows the workstation time which is calculated based on the robot task time. Figure 5.6 shows the final allocation done based on the recursive procedure. And the cycle time obtained from the procedure is 199.

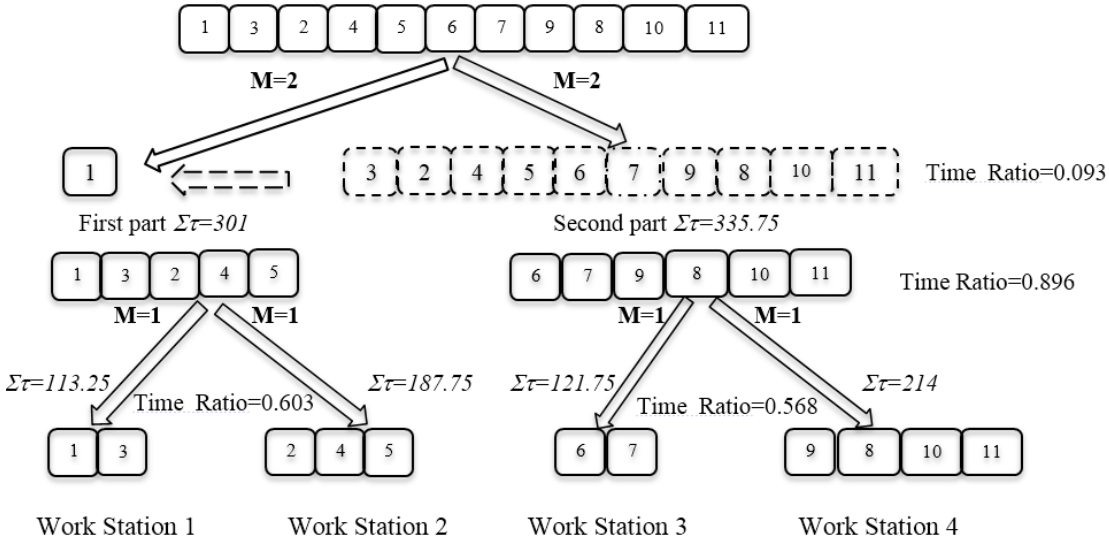

Figure 5.4 Recursive Procedure for splitting tasks to the given workstations

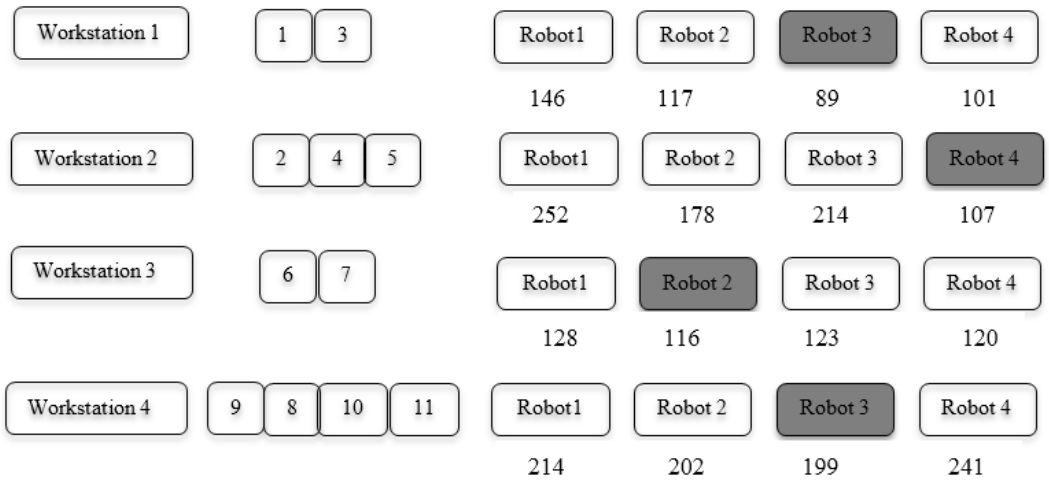

Figure 5.5 Allocation of the best fit robot - recursive allocation procedure

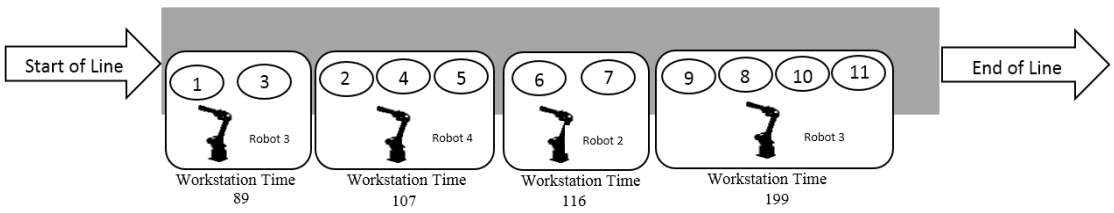

Figure 5.6 Final solution based on recursive allocation procedure

Consecutive Allocation Method (Cycle Time Calculation)

The consecutive allocation method is used for solving RALB problems for the task and robot allocation. This procedure helps in calculating the cycle time for the straight robotic assembly line (Levitin et al., 2006). This heuristic method is used to minimize

the cycle time of an assembly line. This procedure helps in calculating the cycle time for the straight robotic assembly line. An initial cycle time, C_0 is considered for starting the procedure. The procedure aims at assigning tasks and robots consecutively to the workstations in an efficient manner. The procedure allows the assignment of maximum number of tasks to be performed at each workstation by the available robots. C_0 value is i4ncremented by one if all tasks are not possible to be allocated to the workstations. The stepwise procedure of the consecutive heuristic is explained in this section.

Step 1: Initial value of C_0 is the average of the minimum performance time of robots for the tasks. C_0 is calculated as follows:

$$\text{Initial assembly line time } C_0 = \left\lceil \sum_{j=1}^{N_a} \min_{1<i<N_r} t_{i,h} / N_w \right\rceil \qquad \textbf{(5.7)}$$

The robot performance times shown in Table 5.2 are used for the illustration. The following feasible sequence of tasks which meets the precedence constraints is considered for illustration.

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

Initial C_0 for the sample data is calculated and it is 109.

C_0= [37+42+38+40+25+65+40+34+33+41+38]/4=108.25. Here 37,42,38,40,25,65,50,34,33,41,38 are the minimum robot task times (refer Table 5.2).

Step 2: Workstation is opened and tasks are allocated based on the sequence in the order of occurrence, procedure checks if one or more robot could perform the allocated tasks within C_0. For each workstation'*s*' it has a set of preferred/ allotted robots *H* which is defined as follows:

$$k \in H, \; if \;\; m(k) \geq m(h), \;\; \text{where } 1 \leq h \leq N_r \qquad \textbf{(5.8)}$$

Here, *m (h)* is the maximal number of activities a robot *h* can perform in the given sequence *sq* during a time lesser than C_0 .

$$T_s(h) = \sum_{k=p1_s}^{p1_s+m(h)} t_{h,sq(k)} < C_0 \leq \sum_{k=p1_s}^{p1_s+m(h)+1} t_{h,sq}(k) \qquad \textbf{(5.9)}$$

Next, it defines the robot (worker) to be assigned to the workstation *s* as:

$$h(s) = k, \text{ if } T_s(k) < T_s(h) \quad \forall h \in H \qquad \textbf{(5.10)}$$

Step 3: Start position ($p1_{s+1}$) for the next station is calculated

$$p1_{s+1} = pr_w + 1 = p1_s + m(h(s)) + 1 \qquad \textbf{(5.11)}$$

Step 2 is repeated until all tasks are assigned to given number of workstations**.**

Step 4: If all tasks cannot be assigned to the given number of workstations within the initial C_0, C_0 is incremented by 'one' and steps 2 and 3 are repeated until all the tasks gets allotted to the give number of workstations.

Step 5: Robot is assigned to the workstation based on the minimum robot performance time for the allotted tasks.

Step 6: The cycle time of the assembly line is evaluated. The workstation time which is the maximum among all the workstation time is the cycle time of the assembly line.

Figure 5.7 represents the allocation when C_0 is 109; tasks 8, 10 and 11 remain unassigned. C_0 was incremented by one to allocate all the tasks and procedure finds the solution for all the four workstations after C_0 reaches 143 as shown in Figure 5.8. Cycle time of the given sequence is 143 when allocation is done based on consecutive allocation procedure. In Figure 5.8, shaded portion shows the robot which is allotted to the workstation to perform the tasks. Values shown next to the boxes are the task performance time for the corresponding robot. Figure 5.9 shows the final allocation done based on the recursive procedure. And the cycle time obtained from the procedure is 143.

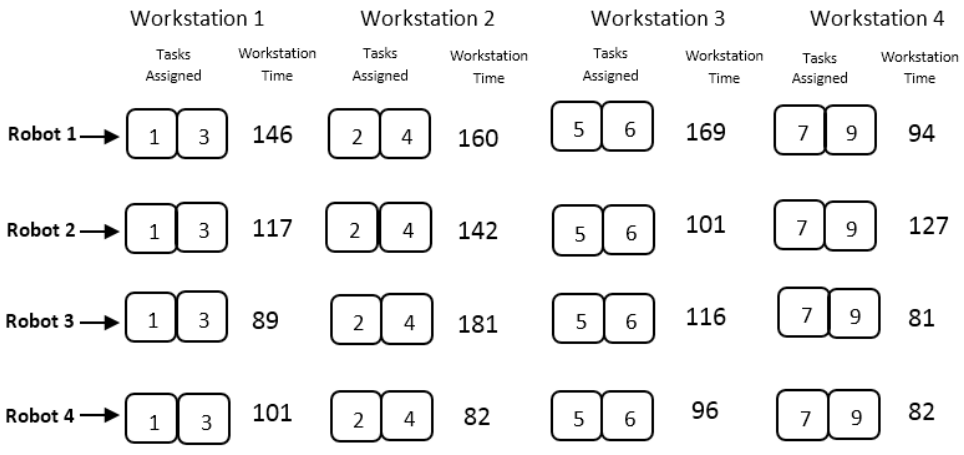

Figure 5.7 Task and Robot allocation - consecutive method with initial C_0

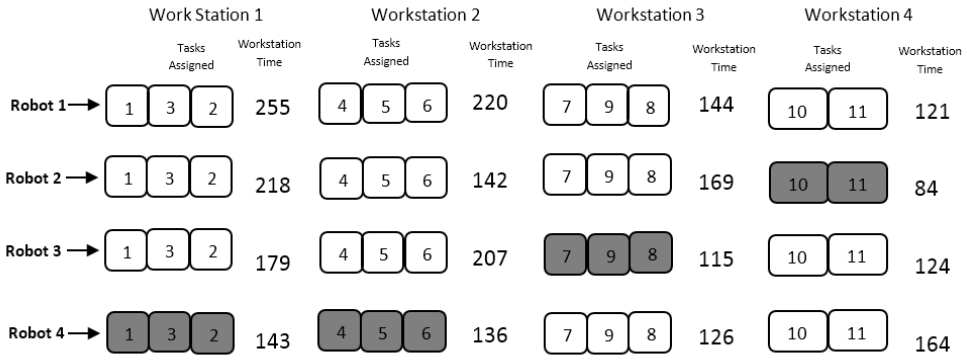

Figure 5.8 Allocation of best fit robot - consecutive allocation procedure

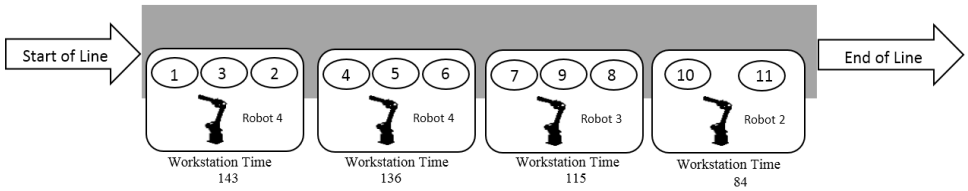

Figure 5.9 Final solution based on consecutive allocation procedure

d) Exchange Procedure (Solution Improvement)

A local exchange procedure is used to improve the quality of the solution. The best solution obtained for all iteration is subjected to the local exchange procedure to find out if any improvement can be achieved. The procedure is explained below:

Step 1: The final global best and final workstation assignments obtained from PSO is used as the input for the local exchange procedure.

Step 2: Procedure finds the station with highest cycle time and tries to shift a task to the adjacent workstations with lower cycle time in such a manner that task added to the adjacent station does not exceed the cycle time of the station from where we removed.

Now consider two workstations f and q with total execution times T_f and T_q such that $T_f > T_q$. If shifting of activities from station f to q is feasible, the new execution time after shifting is as follows:

$$T^*_f = T_f - t_{r,i} \quad \textbf{(5.12)}$$

$$T^*_q = T_q + t_{r,i} \quad \textbf{(5.13)}$$

The exchange is worth-while if

$$\max\{T^*_f, T^*_q\} < T_f \quad \textbf{(5.14)}$$

Step 3: From the solution obtained from the above Step 2, find the station with the lowest cycle time and try to shift a task from the adjacent workstations to it. The workstation *h* with minimum cycle time (T_h) is found and the adjacent workstation *g* such that $T_h < T_g$. If shifting of activities from *g* to *h* is feasible, the new execution time is as follows

$$T^*_g = T_g - t_{r,i} \quad \textbf{(5.15)}$$

$$T^*_h = T_h + t_{r,i} \quad \textbf{(5.16)}$$

The exchange is worth-while if

$$\max\{T^*_g, T^*_h\} < T_g \quad \textbf{(5.17)}$$

This procedure is used in exchange procedure to distribute the tasks amongst the workstations to get a balanced cycle time between them.

Step 4: Repeat step 2 and step 3 until termination condition is satisfied. The termination condition assumed to be 25 iterations.

An example of exchange procedure is explained below in Table 5.10. 25-9 problem of RALB benchmark data is chosen to illustrate the exchange procedure.

The precedence relations of tasks and processing times of the 25 tasks by the 9 robots available in the literature are presented in Table 5.11.

The procedure starts with a cycle time of 127. After performing the Step 2, cycle time is reduced to 125, procedure proceeds with Step 3 by checking for lowest cycle time and tries to allocate tasks to workstation with lower cycle time. Step 2 and Step 3 are repeated for maximum of 25 iterations and cycle time was reduced to 114. Exchange procedure can be performed only if the precedence relationship is not violated.

Table 5.10: Illustration of Local Exchange Procedure

Sequence produced using PSO :
1 2 3 4 8 9 5 6 7 11 12 15 17 23 13 14 16 19 20 21 25 10 24 22 18

Step1: Task and Robot Assigned for the given sequence

Workstations and Tasks	Robot	Cycle Time
S1: 1 2	4	59
S2: 3 4 8	7	117
S3: 9 5 6	7	94
S4: 7 11 12	7	**127**
S5: 15 17 23	4	114
S6: 13 14 16 19	7	125
S7: 20 21 25	8	105
S8: 10 24	4	63
S9: 22 18	7	80

Step 2: Shifting activities from workstations highest C.T.

Workstations and Tasks	Robot	Cycle Time
S1: 1 2	4	**59**
S2: 3 4 8	7	117
S3: 9 5 6 7	7	116
S4: 11 12	7	105
S5: 15 17 23	4	114
S6: 13 14 16 19	7	125
S7: 20 21 25	8	105
S8: 10 24	4	63
S9: 22 18	7	80

Step 3: Shifting activities to workstations with lowest C.T.

Workstations and Tasks	Robot	Cycle Time
S1: 1 2 3	4	114
S2: 4 8	2	81
S3: 9 5 6 7	7	116
S4: 11 12	7	105
S5: 15 17 23	4	114
S6: 13 14 16 19	7	125
S7: 20 21 25	8	105
S8: 10 24	4	63
S9: 22 18	7	80

Step 4: Repeat steps 2 and 3 for 25 iterations to get the solution

Workstations and Tasks	Robot	Cycle Time
S1: 1 2 3	4	114
S2: 4 8 9	7	107
S3: 5 6 7	7	91
S4: 11 12	7	105
S5: 15 17 23	4	**114**
S6: 13 14 16	7	99
S7: 19 20 21	7	100
S8: 2510 24	4	102
S9: 22 18	7	80

Table 5.11 Performance times of 25 tasks for 9 Robots

Task No:	Predecessor Tasks	Performance Time								
		R1	R2	R3	R4	R5	R6	R7	R8	R9
1	-	75	52	52	33	52	63	84	43	38
2	-	108	32	34	26	49	105	28	27	33
3	1,2	135	55	73	55	58	133	35	64	67
4	3	89	47	56	69	57	116	40	87	53
5	4	53	47	48	46	56	84	34	91	50
6	5	55	62	33	26	43	56	35	51	55
7	6	30	33	38	23	30	52	22	37	37
8	4	46	34	77	37	74	47	42	31	28
9	8	62	54	36	43	57	45	25	39	33
10	6,9	52	40	45	37	74	56	41	72	51
11	7,8	111	77	65	71	64	81	57	66	100
12	7	49	34	43	43	58	107	48	60	46
13	9,11	87	32	32	45	34	38	22	40	52
14	13	49	73	46	32	46	49	42	43	69
15	12	64	90	68	39	47	121	72	61	54
16	14	85	128	45	74	44	126	35	64	60
17	15	42	34	31	35	34	40	35	30	28
18	16,17	55	47	95	60	56	55	37	75	69
19	14	56	44	37	51	33	62	26	27	31
20	14	63	61	58	45	67	126	52	44	75
21	20	64	38	32	41	30	34	22	33	34
22	15,19,21	93	106	50	36	106	57	43	84	52
23	17	48	52	45	40	58	49	58	77	59
24	21	58	47	40	26	81	109	29	75	35
25	18,20,23	42	38	39	39	30	40	39	28	32

5.2 PSO variants and Hybrid PSO models for straight RALB

Four different PSO variants and three hybrid PSO models are proposed to solve RALB problems. PSO variants and hybrid models proposed are explained in this section. Four variants of PSO are developed based on the variation in the velocity update equation. And hybrid models are developed based on the hybridized PSO. PSO is hybridized with genetic algorithm and cuckoo search.

5.2.1 PSO variants with inertia weight and constriction factor

Shi and Eberhart (1998) added a new parameter into the original PSO algorithm since the standard version of PSO had no control over the previous velocity of the

particles. Newer version of PSO is incorporated with an inertia weight *w* which addresses the shortcoming of the standard PSO. The inertia weight helps to control the impact of the previous history of velocities on the current one. Inertia weight *w* value can be a positive constant or positive linear or nonlinear function of time. The dynamic equation of PSO with inertia weight is modified to be:

$$v_i^{t+1} = w * v_i^t + c_1 U_1 ({}^e P_i^t - P_i^t) + c_2 U_2 (G - P_i^t) \quad \textbf{(5.18)}$$

Inertia weight helps to balance between global and local search abilities. Large value of inertia weight facilitates global search, while small value facilitates local search. Position update equation remains the same. In this research, this variant of PSO would be referred as PSO-W

Clerc and Kennedy (2002) introduced a parameter called constriction factor (χ) which may help to ensure convergence to the global minimum. A simplified method of incorporating the parameters is given in Equation 5.19, where χ is a function of c_1 and c_2.

$$v_i^{t+1} = \chi * [v_i^t + c_1 U_1 ({}^e P_i^t - P_i^t) + c_2 U_2 (G - P_i^t)] \quad \textbf{(5.19)}$$

$$\chi = \frac{2}{|2 - \phi - \sqrt{\phi(\phi - 4)}|} \quad \textbf{(5.20)}$$

Where, $\varphi = c_1 + c_2, \varphi > 4$.

Here c_1 and c_2 are weight of personal best and weight of global best, respectively; c_1 and c_2 are two positive constant values usually set equal to 2.05 (Eberhart and Shi, 2000) and are also called *cognitive* and *social* parameter, respectively. U_1 and U_2 are random numbers distributed uniformly between 0 and 1. The position of each particle is then updated using Equation 5.2 This variant of PSO will be referred to as PSO-C in this thesis.

5.2.2 PSO variants with time varying inertia weight and constriction factor

Shi and Eberhart (1998) found significant improvement in performance of PSO with the linearly decreasing inertia weight over each iteration, time-varying inertia weight (TVIW) is defined as follows:

$$w = w_{\max} - \frac{w_{\max} - w_{\min}}{\max iter} * iter \quad \textbf{(5.21)}$$

Where w_{max} and w_{min} are the maximum and minimum values of the inertia weight respectively, *iter* is the current iteration number and *maxiter* is the maximum number of allowable iterations. The calculated '*w*' is used in the Equation 5.18 for PSO-W. For population based optimization methods, it is preferred to encourage the individuals to search the entire search space without wandering around the local optima during the initial stages, later stages it is very important to move towards the global optima, to find the optimum solution efficiently. This variant of PSO will be referred to as PSO-TVIW in this thesis.

To address the above mentioned concern, Ratnaweera et al. (2004) proposed time varying acceleration coefficient (TVAC) as a new strategy for the PSO. The method reduces the *cognitive* component (c_1) and increases the *social* component (c_2) of acceleration coefficient, with time. Large value of c_1 and small value of c_2 at the beginning allows the particles to search the whole search space, instead of moving towards the local best. In later stages, values of c_1 need to be small and large values of c_2 helps the particles to converge to the global optima.

Time-varying acceleration coefficient (TVAC) is defined as follows:

$$c_1 = (c_{1i} - c_{1f})\left(\frac{\max iter - iter}{\max iter}\right) + c_{1f} \quad \textbf{(5.22)}$$

$$c_2 = (c_{2i} - c_{2f})\left(\frac{\max iter - iter}{\max iter}\right) + c_{2f} \quad \textbf{(5.23)}$$

Where c_{1i} and c_{2i} are the initial values of the acceleration coefficient c_1 and c_2 and c_{1f} and c_{2f} are the final values of the acceleration coefficient c_1 and c_2, respectively. The calculated c_1 and c_2 are used in Equation 5.19 for PSO-TVAC. This variant of PSO will be referred to as PSO-TVAC in this thesis.

5.2.3 Hybrid PSO variants with inertia weight and constriction factor

Angeline (1998) pointed out that the PSO performs well in the early iterations, but has problems reaching a near optimal solution in several real-valued function optimization problems.

Hybrid models of Genetic algorithm (GA) and PSO helps in improving the quality of the solution obtained (Eberhart and Shi, 1998, Angeline, 1998). Hybrid GA-PSO algorithm basically employs a major aspect of the classical GA approach, which is the capability of "breeding." PSO algorithm has been strengthened using breeding

technique similar to that applied in GA. The pseudo code for the hybrid PSO is presented in Figure 5.10**.**

---

procedure Hybrid PSO
input (problem data, PSO parameters)
begin
$t \leftarrow 0$;
for$(i = 1, N)$
$Generate \;\; P_i^t$;
$Evaluate\ Z\left(P_i^t\right)$;
${}^eP_i^t \leftarrow P_i^t$;
end
$G \leftarrow P_i^t \;\; having\ min\ \{Z\left({}^eP_i^t\right); \;\; i = 1, N \;\}$
for $(i = 1, N)$
$Initialize \;\; v_i^t$;
end
do {
for$(i = 1, N)$
Update Position P_i^{t+1}
Update Velocity v_i^{t+1} (Using Equation 5.18 and 5.19)
end
Breeding
$Evaluate\ all\ particles$
$Update\ {}^eP_i^t \;\; and\ G, (i = 1, N)$
$t \;\leftarrow t + 1$
} (**while** $(t < t_{max})$)
output G (Apply Exchange Procedure for the output)

Figure 5.10 Pseudo Code of Hybrid PSO

However, some researchers have included mutation or simple replacement of the best fitted value, as a means of improvement to the standard PSO formulation (Naka et al., 2003, El-Dib et al., 2004) Lovbjerg *et al*.(2001) shown in their work that PSO hybrid with breeding has the potential to reach a better optimum than the standard PSO. Breeding is included for both the variants of PSO. Velocity update is done based on the Equation 5.18 for HPSO-W and 5.19 for HPSO-C.

The difference in the hybrid PSO is the incorporation of breeding. The implementation of breeding is explained below with an example.

5.2.3.1 Breeding

Step 1: Two particles are selected randomly from the population to perform the breeding process. This particles selected are for 11 task problem of RALB. The particle meets the precedence relationship.

parent₁

1	2	3	6	5	4	7	8	10	9	11

Let the velocity of *parent₁* be: (2, 3) (4, 5) (3, 4) (8, 9)

parent₂

1	2	5	3	6	4	8	7	9	10	11

Let the velocity of *parent₂* be: (1, 2) (3, 4) (2, 4) (3, 5) (8, 9)

Step 2: Generate children for the parents selected using Equations 5.24 and 5.25.

$$child_1(p) = U * parent_1(p) + (1-U) * parent_2(p) \quad \textbf{(5.24)}$$

$$child_2(p) = U * parent_2(p) + (1-U) * parent_1(p) \quad \textbf{(5.25)}$$

For the parents selected, the children generated are,

child₁

1	2	3	6	5	4	8	7	9	10	11

child₂

1	2	5	3	6	4	7	8	10	9	11

Here *U* is assumed to be 0.5. So it means 50% values will be taken from each parent to form child. For *child₁* 50% of values come from first parent and remaining 50% comes from the second parent. For *child₂* 50% of values come from second parent and 50% of remaining values comes from the first parent. A reordering procedure as illustrated by (Levitin et al., 2006) is adopted to repair the infeasible child formed.

Step 3: Velocity Calculation of Child

$$child_1(v) = (parent_1(v) + parent_2(v)) * \frac{|\ parent_1(v)\ |}{|\ parent_1(v) + parent_2(v)\ |} \quad \textbf{(5.26)}$$

$$child_2(v) = (parent_1(v) + parent_2(v)) * \frac{|\ parent_2(v)\ |}{|\ parent_1(v) + parent_2(v)\ |} \quad \textbf{(5.27)}$$

Where, $|parent_1(v)|$ is the number of velocity pairs in $parent_1$ and $|parent_2(v)|$ is the number of velocity pairs of $parent_2$. Velocity of the $child_1$ and $child_2$ are calculated using Equation 5.26 and 5.27 respectively as follows:

Velocity of child$_1$:

= (2,3)(4,5)(3,4)(8,9)(1,2)(2,4)(3,5) * (4/9) = (2,3)(4,5)(3,4)(8,9)(1,2)(2,4)(3,5) * 0.4

= (2, 3) (4, 5) (3, 4)

Velocity of child$_2$:

= (2,3)(4,5)(3,4)(8,9)(1,2)(2,4)(3,5) * (5/9) =(2,3)(4,5)(3,4)(8,9)(1,2)(2,4)(3,5) * 0.5

= (2,3)(4,5)(3,4)(8,9)

(Note: the ratio (5/9) 0.5 is used randomly to inherit 50% of velocity pairs for the child.)

Step 4: The parent particles are replaced by their child particles, thereby keeping the population size fixed if the fitness value of the child is better than the parent otherwise parents are retained.

5.2.4 Hybrid Cuckoo Search-PSO variant

Recently, Yang and Deb (2009) proposed a new metaheuristic algorithm called cuckoo search (CS) because of the obligate brood parasitism of some cuckoo species by laying their eggs in the nests of other host birds (of other species). CS algorithm has shown good performance for solving both benchmark unconstrained functions and real-world problems in manufacturing scheduling (Burnwal and Deb, 2013, Long et al., 2014). The CS is based on three idealized rules (Yang and Deb, 2009):

- Each cuckoo lays one egg at a time, and dumps it in a randomly chosen host nest;
- The best nests with high quality of eggs (solutions) will carry over to the next generations;
- The number of available host nests is fixed, and a host can discover an alien egg with a probability p_a. In this case, the host bird can either throw the egg away or abandon the nest to build a completely new nest in a new location.

In this section, a hybrid CS-PSO algorithm is proposed to solve RALB problem. The pseudo code of the hybrid CS-PSO is given in the Figure 5.11.

```
begin
    Generate initial a population of n host nests, x_i (i=1,2,...n);
    Evaluate objective function x=(x_1,...x_d)^T;
     while(t<MaxGeneration) or (stop criterion);
       Randomly select two cuckoos in the population;
       Generate a new cuckoo using OX operator;
       Evaluate the new cuckoo and record the local best (F_i);
   Choose a nest among (say j) randomly;
       if (Fi < F j)
       Replace j by the new solution;
       end
       Move cuckoo birds using Equation 5.1 and 5.2;
       Abandon a fraction (pa) of worse nests;
       Build new ones at new locations using single point crossover;
       Rank the solutions and find the current best;
    end while
     Record the global best
end
```

Figure 5.11 Pseudo code of hybrid CS-PSO algorithm

Nature of cuckoo birds is that they do not raise their own eggs and never build their own nests, instead they lay their eggs in the nest of other host birds. If the alien egg is discovered by the host bird, it will either throw these alien eggs away or simply abandon its nest and build a new nest elsewhere. Thus, cuckoo birds always look for a better place in order to decrease the chance of their eggs to be discovered (Burnwal and Deb, 2013). In the hybridized algorithm, communication for cuckoo birds is incorporated by hybridizing CS with PSO. Motive of this is to communication and inform each cuckoo bird to migrate to a better place. Each cuckoo bird will record the best personal experience as local best (${}^{e}p^{t}_{i}$) during its own life. In addition to this, the local among all the birds called Global best (G^{t}) is also recorded. The cuckoo birds' communication is established through the local, global best and they update their position and velocity using these parameters using Equation 5.1 and 5.2 of the PSO algorithm.

5.2.4.1 New Cuckoo Generation

New cuckoos are generated by using Order crossover (OX) operator which is proposed by Davis (1985). In case of RALB problem, the sequence (cuckoo) is the group of tasks arranged in such a way that it satisfies the precedence conditions. Working of the OX operator is explained below:

- Select a subsection of task sequence from one parent at random.

- Produce a proto-child by copying the substring of task sequence into the corresponding positions.
- Delete the tasks that are already in the substring from the second parent. The resulted sequence of tasks contains tasks that the proto-child needs.
- Place the tasks into the unfixed positions of the proto-child from left to right according to the order of the sequence in the second parent.

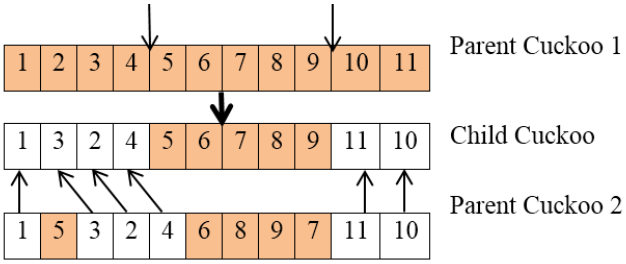

Figure 5.12 Illustration of the OX operator

An illustration is presented in Figure 5.12. Two cuckoos from the population are selected and OX operator is applied to generate a new cuckoo. The new child cuckoo generated is shown in the figure.

5.2.4.2 Replacement of abandoned cuckoos

New cuckoos are generated from the abandoned cuckoos using Single Point Cross over method. It is a cross over method where a single crossover point on both parent strings is selected. All data beyond that point is swapped between the two parent organisms. The resulting sequences are the children. An illustration is presented in Figure 5.13.

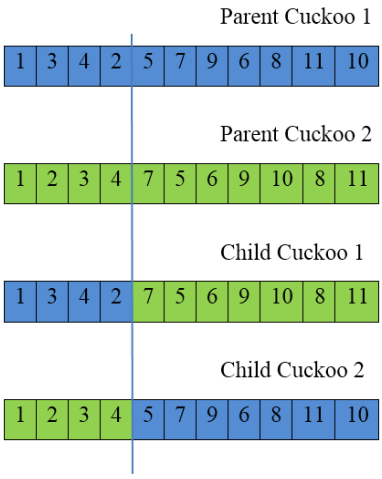

Figure 5.13 Illustration of the Single Point Cross over

5.2.4.3 Repair mechanism

If the child cuckoos created using the crossover methods is not meeting the precedence constraints, the cuckoo is repaired using a re-ordering procedure which restores the feasibility of the generated sequence according to the precedence constraints. Detailed description of the reordering procedure is illustrated by Rubinovitz and Levitin (1995).

5.3 Experimental and Computational Study for Standard PSO

The results of computational analysis for different metaheuristic algorithms are presented in this section

5.3.1 Parameter Selection for standard PSO

Performance of PSO mainly depends on the parameters used. Extensive experiments are done to find the best optimal parameters. Initially three data sets are chosen to find the parameters which produces good solution. The three problems selected are 35 tasks-12 robots, 70 tasks- 19 robots and 148 tasks-29 robots problems. These problems are solved for all combination of the parameters for 10 test runs. Quality of solution is given importance compared to the computational time. The details of the analysis conducted for both the allocation methods (recursive and consecutive methods) and the parameters chosen for the proposed PSO is explained in this section.

Stopping condition: The proposed two methods are terminated if the iteration approaches a predefined criteria, usually a sufficiently good fitness or in this case, a predefined maximum number of iterations (generations) is used. Different stopping conditions are tested such as 5, 10, 15, 25 and 30 and best solution is obtained when number of generation is 25, which is shown in Figure 5.14 for recursive allocation procedure and Figure 5.15 for consecutive allocation procedure. From the Figure 5.14 it is observed that for both the allocation methods procedure starts producing the same solution for most of the runs after 25^{th} iteration. Values shown in Figure 5.14 represent the cycle time obtained for different stopping condition.

Acceleration coefficients: c_1, c_2 and c_3 are assumed to have the value of 1 or 2. Various combinations of c_1, c_2 and c_3 with 1 and 2 are tried to find the best possible combination. The complete set of combinations of acceleration coefficients tested for

both recursive and consecutive allocation method are shown in Table 5.12. Performances of the algorithms are tested on three problems for finding the best combination of acceleration coefficients. Figure 5.16 shows the performance of proposed recursive allocation procedure for different combination of acceleration coefficients. Group B with $c_1=1$, $c_2=1$ and $c_3=2$ produces good results for the three problems presented here. Same combination of acceleration coefficients is used for evaluating all the problems available in the literature. Figure 5.17 shows the performance of proposed consecutive allocation procedure for different combination of acceleration coefficients. From Figure 5.17, Group D with c_1=1, c_2=2 and c_3=2 has the best combination of acceleration coefficients to get the minimum cycle time for all the three problems considered. For solving all the problems from the literature, acceleration coefficients from Group D is used.

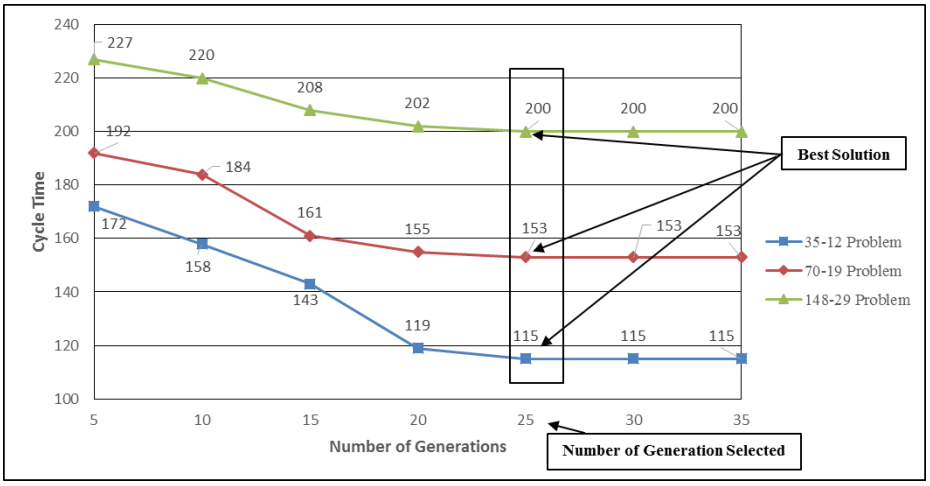

Figure 5.14 Performance of PSO for stopping condition of recursive allocation procedure

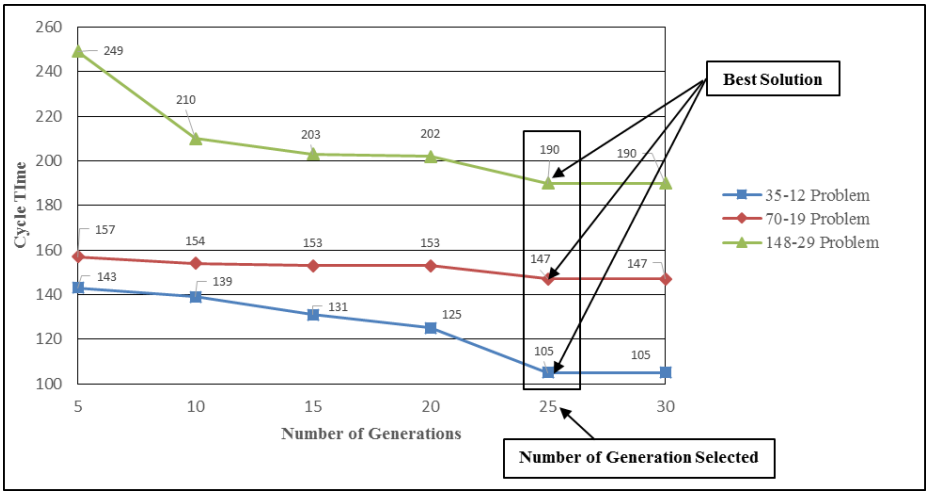

Figure 5.15 Performance of PSO for stopping condition of consecutive allocation procedure

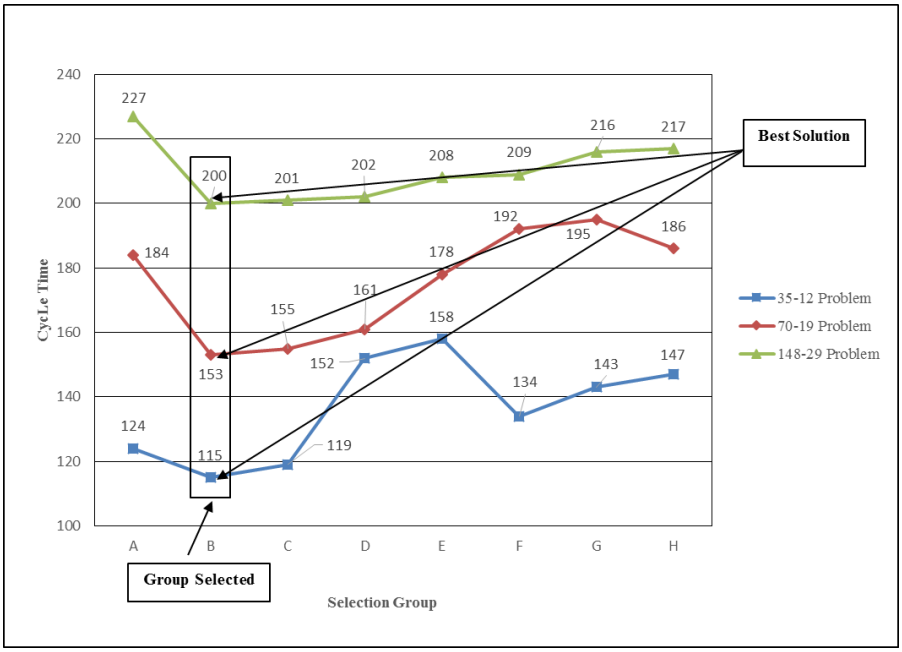

Figure 5.16 Performance of PSO for acceleration coefficients of recursive allocation procedure

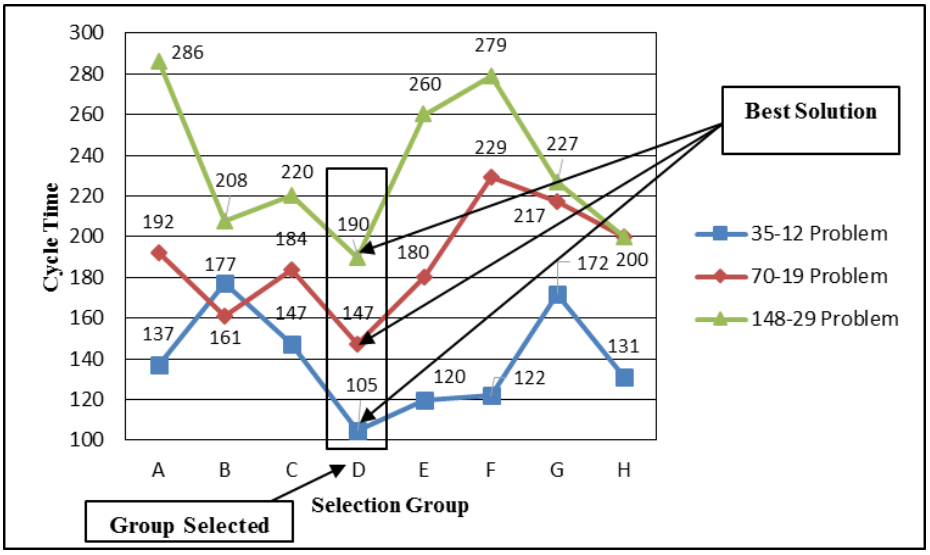

Figure 5.17 Performance of PSO for acceleration coefficients of consecutive allocation procedure

Population Size: Different ranges of population size of the swarm (initial particles) are tested and from the analysis done, best solution is obtained when the population size is 25 for both recursive and consecutive algorithm. Figure 5.18 shows the performance of PSO for different population size for three problems for recursive allocation procedure and Figure 5.19 shows the performance of consecutive allocation procedure for the three sample problems with different population size. It is analyzed that, when the population size increases the quality of the solution increases.

Table 5.12 Selection of c_1, c_2 and c_3

Acceleration Coefficients			
Group	c_1	c_2	c_3
A	1	1	1
B	1	1	2
C	1	2	1
D	1	2	2
E	2	1	1
F	2	1	2
G	2	2	1
H	2	2	2

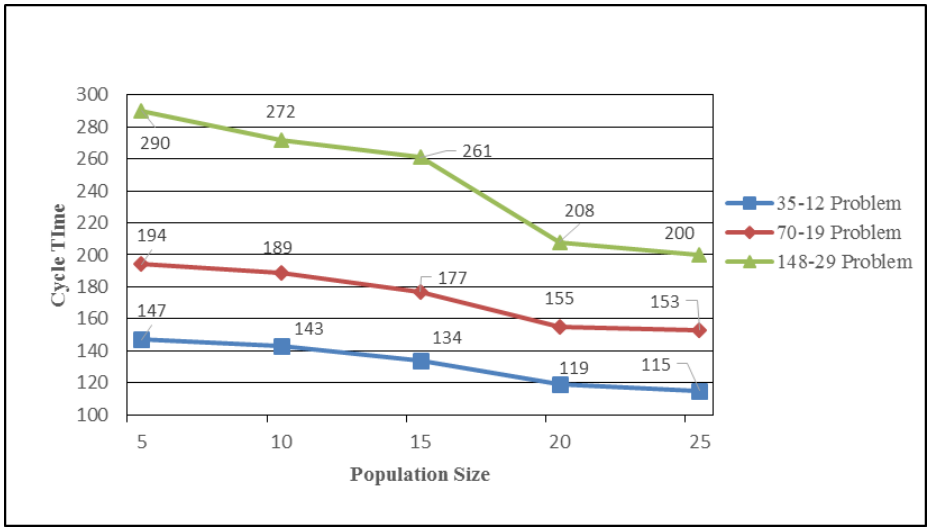

Figure 5.18 Performance of PSO for population size of recursive allocation procedure

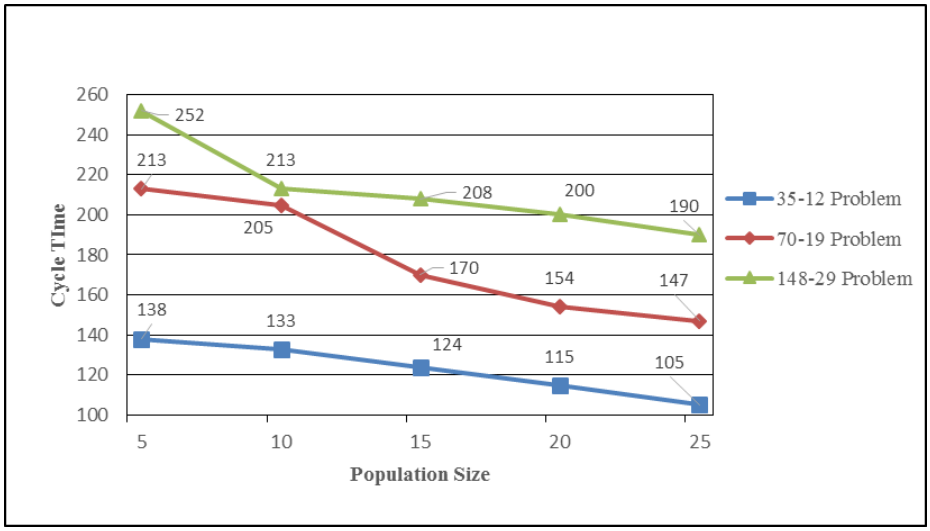

Figure 5.19 Performance of PSO for population size of consecutive allocation procedure

5.3.2 Computational Study for Standard PSO

The computational experiments are conducted in order to test the performance of the proposed standard PSO on RALB-2 problems. The details of the experiments conducted are presented in this section. Large set of benchmark problems are evaluated to check the performance of the proposed algorithms. Precedence graphs which are widely used in SALB-I literature (Scholl, 1995) are selected from http://www.assembly-line-balancin.de/ to perform the experimental study. Gao et al. (2009) generated 32 test problems for the computational study of RALB-2 problems by

adding the robot task times to the original datasets prepared by Scholl (1995). Problem size varies between 25 and 297 tasks. Eight different problems available in the literature with different combination of robot problems are considered for the computational study. Table 5.13 presents the details of the 8 problems considered and the data's are collected from various industries used by earlier researchers. The procedure adopted for developing the datasets are presented in their work. The two proposed (recursive and consecutive) algorithms are evaluated using the 32 benchmark problems available in the literature. For each test problem, the number of workstations is equal to the number of robots and only one robot of particular type can be allocated to one workstation. Each task is assigned to one workstation and the robot type are selected for that workstation without violating precedence constraints, and the processing time of a task at the assigned station is dependent on the type of robot selected for that workstation. In this problem formulation, assignment of robot type differs from Gao et al.'s model. Gao et al. (2009) confined only to one robot is available in each robot type. Though their model differs in one aspect, the solution of hybrid GA (Gao et al., 2009) can be used to fix the upper bound for the problem under consideration.

Table 5.13 Source of Datasets

Problem Size	Source	Reference
25	Randomly Generated	(Rosenberg and Ziegler, 1992)
35	Assembly of an auto engineer cradle	(Gunther et al., 1983)
53	Assembly of Hot Tank	(Hahn, 1972)
70	Electronic Industry	(Tonge, 1960)
89	Assembly of Refrigerator	(Lutz, 1974)
111	Mixed Assembly Line	(Arcus, 1965)
148	Assembly of small Utility Vehicles	(Bartholdi, 1993)
297	Assembly of an Engine	(Scholl, 1999)

The solution to the RALB problem includes an attempt for optimal assignment of robots to line stations and a balanced distribution of work between different stations. The results obtained by evaluating 32 test problems for the objective of minimizing cycle time in a straight robotic assembly line balancing (RALB-2) problems are presented in Table 5.14. Column I shows the problem number. Column II shows the task size of the problems evaluated and Column III shows the number of workstations/robots for each problem. Column IV shows the WEST ratio of the problem (explained in the next section) and V shows the cycle time for the 32 test problems, Column VI show computational time and Column VII shows the modified

cycle time for hGA algorithm. Results obtained from Cplex (optimization solver) are presented in Column VIII and IX. Column X and XI shows the cycle time and computational time obtained using the proposed PSO for recursive allocation procedure. Results obtained from proposed PSO for consecutive allocation procedure is presented in Column XII and XIII.

IBM Cplex Optimization studio Version 12.6.0.0 is used to solve the problems. All 32 problems could not generate solution using Cplex. Since, the objective function of the formulation for this problem is non-linear and hence it is hard for traditional exact optimization techniques to solve the problems. Cplex could generate the solutions for the first fourteen problems and the optimization solver displays an error message as, 'Search Space Exceeded'. Results obtained by the proposed algorithms are compared with results obtained using hybrid GA (upper bound for the problem) (Gao et al., 2009) and Cplex solver solutions. The non-deterministic nature of the algorithm and problem makes it necessary to run same problem multiple times. Each problem is run ten times and most of the runs converged to the same solution for each of the problems. It is found that the results of PSO using both allocation procedures are very close to Cplex solutions and PSO with Consecutive procedure yields better results when compared to PSO with recursive and hGA.

Table 5.14 Results of the 32 straight RALB-2 problems

Problem Number	No. of Tasks	Work Station/Robots	WEST Ratio	Hybrid GA (Gao et al., 2009)			Cplex		PSO-Recursive Algorithm		PSO-Consecutive Algorithm	
				Cycle Time	CPU Time (sec)	Modified CPU Time (sec)	Cycle Time	CPU Time (sec)	Cycle Time	CPU Time (sec)	Cycle Time	CPU Time (sec)
1	25	3	8.33	503	5	2	503	0.5	503	3.5	503	2.65
2		4	6.25	327	5	2	292	8.85	327	3.9	327	2.9
3		6	4.17	213	6	2	200	463	208	4.2	208	3.0
4		9	2.78	123	7	3	109	4002	114	4.8	114	3.25
5	35	4	8.75	449	12	5	341	59	347	7.7	344	4.9
6		5	7.00	344	16	7	329	562	338	7.9	336	5.4
7		7	5.00	222	22	9	201	3672	219	7.9	214	6.9
8		12	2.92	113	30	12	106	3620	115	8.3	105	8.5
9	53	5	10.60	554	17	7	449	72	538	22.0	454	13.1
10		7	7.57	320	21	9	283	3680	304	22.4	301	14.9
11		10	5.30	230	27	11	221	3627	228	22.7	224	16.2
12		14	3.39	162	35	14	142	3729	153	22.9	146	19.9
13	70	7	10.00	449	40	16	394	4007	448	46.4	431	29.0
14		10	7.00	272	53	22	245	3700	266	47.3	269	32.5
15		14	5.00	204	64	26	N/A	N/A	204	47.8	200	39.1
16		19	3.68	154	82	33	N/A	N/A	153	48.2	147	43.4
17	89	8	11.13	494	46	19	N/A	N/A	479	88.7	463	41.9
18		12	7.42	370	58	24	N/A	N/A	345	89.4	355	50.4
19		16	5.56	236	71	29	N/A	N/A	234	92.1	234	59.6
20		21	4.24	205	89	36	N/A	N/A	201	93.2	176	75.3
21	111	9	12.33	557	157	64	N/A	N/A	551	167.2	526	82.3
22		13	8.54	319	192	78	N/A	N/A	316	167.6	316	89.5
23		17	6.53	257	229	93	N/A	N/A	257	168.2	254	98.5
24		22	5.05	192	271	110	N/A	N/A	190	171.2	185	110.8
25	148	10	14.80	600	240	98	N/A	N/A	593	381.1	603	179.8
26		14	10.57	427	297	121	N/A	N/A	426	385.5	420	205.5
27		21	7.05	300	332	135	N/A	N/A	299	390.4	277	215.9
28		29	5.10	202	417	169	N/A	N/A	200	490.9	190	230.3
29	297	19	15.63	646	824	335	N/A	N/A	767	1123.5	608	891.8
30		29	10.24	430	907	369	N/A	N/A	451	1235.3	397	997.6
31		38	7.82	344	996	405	N/A	N/A	350	1365.2	295	1269.9
32		50	5.94	256	1103	448	N/A	N/A	257	1392.3	245	1390.8

Results in Table 5.14 shows that the proposed approaches are quite efficient with PSO algorithm to find out the best solutions for almost all data sets. Both recursive and consecutive procedures converged to same set of solution for the first four problems. However the proposed PSO with recursive procedure could not obtain better results for large dataset problem (297 task problems), whereas PSO with consecutive procedure yields better results for the entire 32 test problems.

The computational time (CPU Time) taken for PSO algorithm with recursive allocation procedure is more when compared to PSO with consecutive allocation

procedure for all the thirty two datasets but lesser than hybrid GA for 17 out of 32 problems. Whereas, computation time for PSO with consecutive procedure is lower than that of PSO with recursive procedure and hGA for all datasets. But PSO with consecutive procedure takes more time for the large size datasets problems (297 task problems). This could be due to the size of the problem and more time is taken for the exchange procedure (local search). hGA and PSO are coded on different computers, hence it is difficult to compare the computational times. An approximate comparison of CPU execution times is done using Passmark Performance Test 8.0 software. The hybrid GA (hGA) algorithm is executed on a Pentium 4 processor (2.6-GHz).The proposed PSO algorithms are coded in C++ and are tested on Intel core i5 processor (2.3 GHz). Using the Passmark Performance Test 8.0 software, the factor for the computer used to solve PSO algorithms is fixed to 1 and for the computer used to solve hGA is found to be 0.406. Since there are too many factors affecting the CPU times it is difficult to do a fair comparison. In Table 5.14, column VII gives the modified CPU time of the average best solution computational time. From the table it is found that modified CPU time for hGA is lower than that of the proposed algorithms for most of the datasets. This could be due to large solution space and the local exchange procedure used. However for the proposed algorithms, quality of the solution is given importance compared to the computational time. Figure 5.20 shows that the average CPU time for the proposed PSO procedures are comparable to hGA for the problems up to 111 task problems. Average CPU times for the proposed PSO procedures are comparable to hGA for the problems up to 111 task problems. Table 5.15 shows the percentage deviation for both PSO procedures from the benchmark results. It is found that recursive allocation procedure with PSO is able to produce results with an average reduction of 1.35% and PSO with consecutive allocation procedure performs well and reduces the objective function with an average reduction of 5.55%.

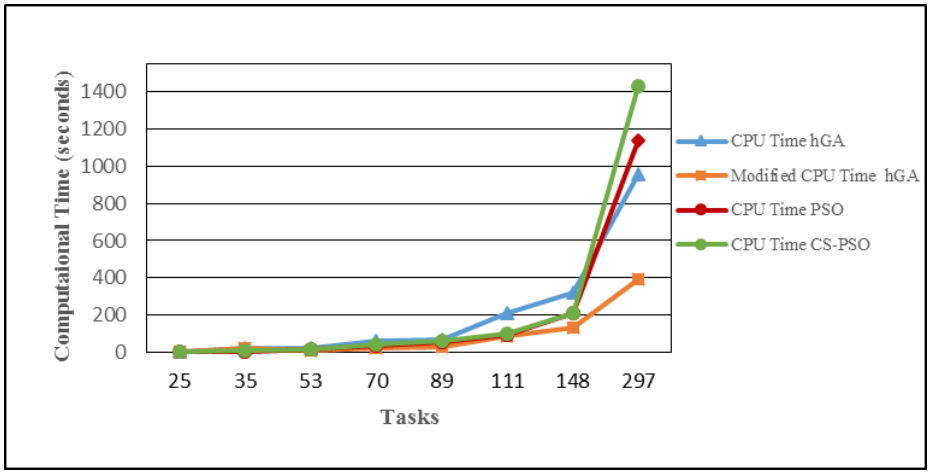

Figure 5.20 Comparison of Average Computational Time

Table 5.15 Percentage Deviation of cycle time for PSO with recursive and consecutive procedure

Problem No:	Problem Dataset	% Deviation-Recursive	% Deviation-Consecutive		Problem No:	Problem Dataset	% Deviation-Recursive	% Deviation-Consecutive
1	25-3	0	0		17	8	-3.04	-6.28
2	25-4	0	0		18	12	-6.76	-4.05
3	25-6	-2.34	-2.34		19	16	-0.85	-0.85
4	25-9	-7.32	-7.32		20	21	-1.95	-14.15
5	35-4	-22.72	-23.39		21	9	-1.08	-5.57
6	35-5	-1.74	-2.33		22	13	-0.94	-0.94
7	35-7	-1.35	-3.6		23	17	0	-1.17
8	35-12	1.77	-7.08		24	22	-1.04	-3.65
9	53-5	-2.89	-18.05		25	10	-1.17	0.5
10	53-7	-5.0	-5.94		26	14	-0.23	-1.64
11	53-10	-0.87	-2.61		27	21	-0.33	-7.67
12	53-14	-5.56	-9.88		28	29	-0.99	-5.94
13	70-7	-0.22	-4.01		29	19	18.73	-5.88
14	70-10	-2.21	-1.1		30	29	4.88	-7.67
15	70-14	0	-1.96		31	38	1.74	-14.24
16	70-19	-.0.65	-4.55		32	50	0.39	-4.3
					Average Reduction (%)		**-1.36**	**-5.5**

5.3.3 Complexity of the problem

Computational complexity of the simple ALB is known to be a non-deterministic polynomial time hard (NP-hard) problem (Karp, 1972). Faaland et al. (1992) stated that the procedures which attempts to find optimal solution would have a complexity of at least 2^N. Due to the complexity of the problem, to solve problems of practical size metaheuristic remains the only option for the researchers. Different measures are

reported in the literature which is used to show the complexity of RALB. F-ratio proposed by Mansoor and Yadin (1971)**,** WEST ratio proposed by Dar-El (1973) and relative complexity proposed by Bhattacharjee and Sahu (1990) are used in this thesis to show the complexity of RALB problems. Simple assembly line balancing problem falls under the category of NP-hard (Gutjahr and Nemhauser, 1964). RALB problem is further complicated with addition of different robot types. Hence, RALB is also NP-hard.

Total nodes are to be calculated to measure the complexity of the given RALB problem. Total nodes measures the computational time required to reach a solution, counting the total number of nodes generated in the search process. Total number of nodes of the problem is directly proportional to the number of iterations in the algorithm, and hence, the computational time (Rubinovitz et al., 1993). The following parameters were used to characterize the RALB problem complexity:

1. Assembly Flexibility- F- Ratio measures the flexibility in creating assembly sequences developed by Mansoor and Yadin (1971) and defined as follows: Let P_{ij} be an element of a precedence matrix P, such that:

$$P_{ij} = \begin{cases} 1 \ \textit{if} \ \text{task } i \ \text{precedes task } j \ \text{is assigned to robot } h \ \text{in station } s \\ 0, \ \textit{otherwise} \end{cases}$$

Then, F-ratio is calculated as follows: $F\text{-}Ratio = 2\text{x}Z / (N_a \text{x}(N_a - 1))$ where Z is the number of zeroes in P, and N_a is the number of assembly tasks. F-ratio value is therefore between zero and one. When there are no precedence constraints between tasks (any sequence is feasible) F-ratio is zero and one when only a single assembly sequence is feasible. Assembly tasks are often characterized by relatively low F-ratios. Problems with eight levels of F-ratio are generated and evaluated. Figure 5.21 shows F-ratio versus Computational time for 8 problems sets for two allocation procedures. Computational time is presented in Table 5.14. It is noted that computational time is an increasing function of F-ratio. A high F-ratio indicates that there are fewer alternatives for assigning tasks to workstations where as low F-ratio gives different ways of assignment**.** Complexity of the line balancing problem depends on F-ratio (Bhattacharjee and Sahu, 1990). From Figure 5.21 it is observed that consecutive allocation procedure solves the problems in a significantly shorter time. In

summary, consecutive allocation procedure performs better in terms of computational time and quality of the solution is superior.

2. WEST ratio- Defined by Dar-El (1973) measures the average number of tasks per workstation. This measure shows the expected quality of achievable solutions and complexity of the problem. (Gao et al., 2009) generated WEST ratios ranging from 2 to 15 to generated 32 RALB problems. For each problem, the number of workstations is equal to the number of robots, and every task can be processed on any robot. WEST ratio considered in this research is shown in Table 5.14.

3. Relative Complexity- $R= (V-Q)/Q$, where V= Computational time for the solution of the problem whose complexity is to be measured and Q= minimum Computational time for the set of the problems under study (Bhattacharjee and Sahu, 1990). Table 5.16 shows the relative complexity for the 32 test problems evaluated using recursive and consecutive procedure for straight robotic assembly line. The relative complexity significantly increases when the problem size increases for both the procedures. However, recursive allocation procedure relative complexity is higher than that of consecutive due to higher computational time.

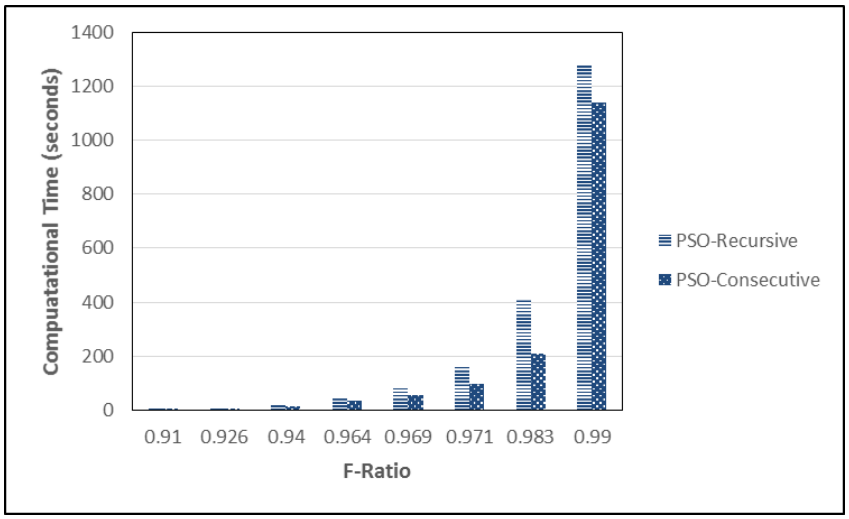

Figure 5.21 F ratio vs computational time

Table 5.16 Relative Complexity of Recursive and Consecutive PSO procedures

Problem No:	Problem Dataset	% Deviation-Recursive	% Deviation-Consecutive	Problem No:	Problem Dataset	% Deviation-Recursive	% Deviation-Consecutive
1	25-3	0	0	17	89-8	25.34	15.811
2	25-4	1.11	1.09	18	89-12	25.54	19.01
3	25-6	1.20	1.13	19	89-16	26.31	22.49
4	25-9	1.37	1.22	20	89-21	26.62	28.41
5	35-4	2.2	1.84	21	111-9	47.77	31.05
6	35-5	2.25	2.03	22	111-13	47.88	33.77
7	35-7	2.25	2.60	23	111-17	48.05	37.16
8	35-12	2.37	3.20	24	111-22	48.91	41.811
9	53-5	6.28	4.94	25	148-10	108.88	67.84
10	53-7	6.40	5.62	26	148-14	110.142	77.54
11	53-10	6.48	6.11	27	148-21	111.54	81.47
12	53-14	6.54	7.50	28	148-29	140.25	86.90
13	70-7	13.25	10.94	29	297-19	321.0	336.52
14	70-10	13.51	12.26	30	297-29	352.94	376.45
15	70-14	13.65	14.75	31	297-38	390.05	479.20
16	70-19	13.77	16.37	32	297-50	397.8	524.83

In summary, it is noteworthy that the proposed consecutive allocation procedure was giving promising results in terms of quality of solution and computational time. Hence, proposed PSO variants and hybrid PSO algorithms for RALB are solved only using consecutive allocation procedure. Following sections presents the details of parameters used for the PSO variants and hybrid models.

5.3.4 Parametric study on PSO variants and hybrid PSO models

The performance of the PSO variants and hybrid PSO models are generally affected by the use of parameters. Extensive experiments and tuning are conducted to find the optimal parameters on all the variants. Three problems of different characteristics are chosen to find the parameters which yielded best solution. Different combinations of the parameters are tested until the best combination is achieved. Solution quality is given importance compared to the computational time in selecting the parameters chosen for all the variants are explained in this section

5.3.4.1 Parameters for PSO –W and PSO-C

Inertia weight *w*, in Equation 5.18, is considered to be critical for the PSO performance. Shi and Eberhart (1998) initially investigated the characteristics when the *w* values range between 0 and 1. Large value of inertia weight helps to search in the

new areas while small inertia weight will facilitate local search i.e. searching in the present area. According to Eberhart and Shi (2000), the optimal strategy is to set *w* to 0.9 initially and then reduce it linearly to 0.4. In this research, inertia weight is set 0.6 for all the generations. Motivation of this concept to select a value in the mid-range is to explore the solution space at an earlier stage and to converge to the solution at faster speed. Three different problems of different characteristics are tested with different values of inertia weight and it is found that when inertia weight was set as 0.6, the best solution is achieved. Figure 5.22 shows the performance variation when different ranges of inertia weight are tested. When inertia weight is set as 0.6, the procedure finds the best solution. For c_1 and c_2 (acceleration coefficients), a set of trial and error run with different range is performed for *PSO-W*. Table 5.17 shows the set of values tested for the selection of c_1 and c_2. After conducting experiments with different combination of parameters it is found that the best solution is obtained when c_1=1 and c_2=2 (Group B). Figure 5.23 shows the performance of PSO-W variant based on the acceleration coefficients. Since the original PSO was tested with 25 generations, PSO-W was tested with 25 generations and results are reported based on these parameters.

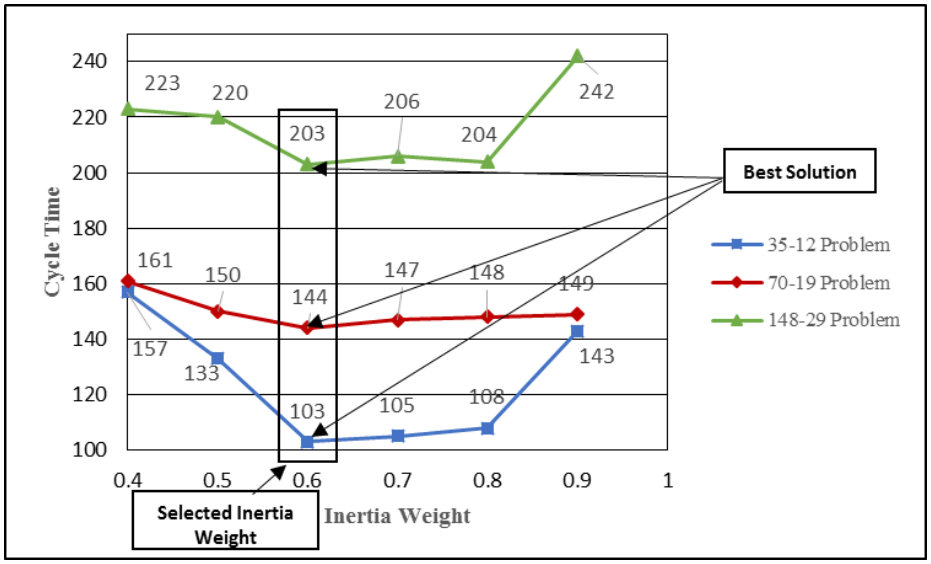

Figure 5.22 Performance variations when different inertia weights are used for PSO-W variant

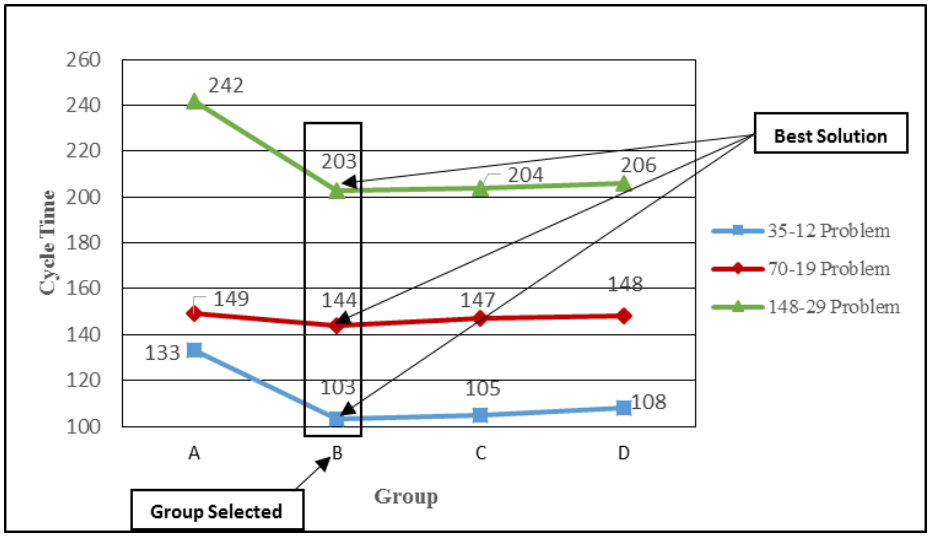

Figure 5.23 Performance variations when different groups of acceleration coefficients are used for PSO-W variant

Table 5.17 Selection of c_1 and c_2

Acceleration Coefficients		
Group	c_1	c_2
A	1	1
B	1	1
C	1	2
D	1	2

Typically, when *PSO-C* method is used, ϕ is set to 4.1 and the χ is thus 0.729. Proper fine-tuning of the parameters c_1 and c_2 in Eq. (18) may result in faster convergence of the algorithm and alleviation of the local minima. The parameters U_1 and U_2 are used to maintain the diversity of the population and are randomly generated values between zero and one and it varies iteratively. Ratnaweera et al. (2004) reported that it will be better to choose a larger cognitive parameter, c_1, than a social parameter, c_2, but with $c_1 + c_2 \leq 4$. Hence in this research, c_1 is set to 2.4 and c_2 to be 1.7. PSO-C is also tested with 25 iterations for all the problems.

5.3.4.2 Parameters for PSO –TVIW and PSO-TVAC

Linearly decreasing inertia weight over the generation has found significant improvement in performance of PSO (Shi and Eberhart, 1998). Parameters for this variant are chosen based on the parameters reported in (Marinakis and Marinaki, 2010). Parameters chosen for PSO-TVIW are as follows: w_{max}=0.9, w_{min}=0.4. c_1 and c_2 chosen

for this variant is same as the one chosen for PSO-W (c_1=1 and c_2=2). The maximum number of iteration is set to 25 (*maxiter*).

For PSO-TVAC, parameters are chosen based on the parameters reported in (Arumugam, 2008). Parameters chosen for this variant are as follows: c_{1i}=2.5, c_{1f}=0.5, c_{2i}=0.5, c_{2f} =2.5. Based on the values calculated for c_1 and c_2 using Equation 5.22 and 5.23 in all iterations, constriction factor is calculated and velocity is updated by using Equation 5.19. For PSO-TVAC variant the maximum number of iteration is set 25.

5.3.4.3 Parameters for HPSO-W and HPSO-C

Different combinations of parameters are tested for HPSO-W. The inertia weight *w* is chosen as 0.6. c_1 and c_2 are also varied from 1 to 2. Best combination for c_1 and c_2 are 1 and 2 same as that of PSO-W. Quality of solution is given importance compared to the computational time. This variant of PSO is an extension of PSO-W, where PSO-W is incorporated with breeding concept of GA. Total number of iterations selected is 25.

PSO-C incorporated with breeding is the variation in this variant, HPSO-C. Most important parameter in this method is the constriction factor. Same constriction factor used for PSO-C is used for this variant. To set the constriction factor as 0.729, c_1 is set to 2.4 and c_2 to be 1.7. Problems are tested for 25 iterations same as PSO-C.

5.3.4.4 Parameters for hybrid CS-PSO

The parameter setting of hybrid CS-PSO algorithm is described in this section. Fraction of worst nests chosen (p_a) is fixed as a constant for all generations in the traditional cuckoo search algorithm. Dynamic p_a is incorporated in this hybrid algorithm. The probability is dynamically updated (Valian et al., 2011) using Equation 5.28

$$p_a(gn) = p_{a\max} - \frac{gn}{N}(p_{a\max} - p_{a\min}) \qquad \textbf{(5.28)}$$

Where N and gn are the number of total iterations and the current iteration respectively.

It is found that p_{amax} =0.8 and p_{amin}=0.1 generates better results. The population size, acceleration coefficients are the same as explained in Section 5.3.1. The proposed method is terminated if the iteration approaches a predefined criteria, usually a

sufficiently good fitness or in this case, a predefined maximum number of iterations (generations) is used. When the number of iterations was set to 25, quality of solution was not satisfactory, hence different stopping conditions are tested such as 5, 10, 15, 25, 30 and 35 to choose the best iteration. Best solution is obtained when number of generation is 30. It is observed that after 30th iteration the procedure produces the same solution for most of the runs. Figure 5.24 shows the performance of algorithm based on the number of generations for three problems (35-12, 70-19, 148-29).

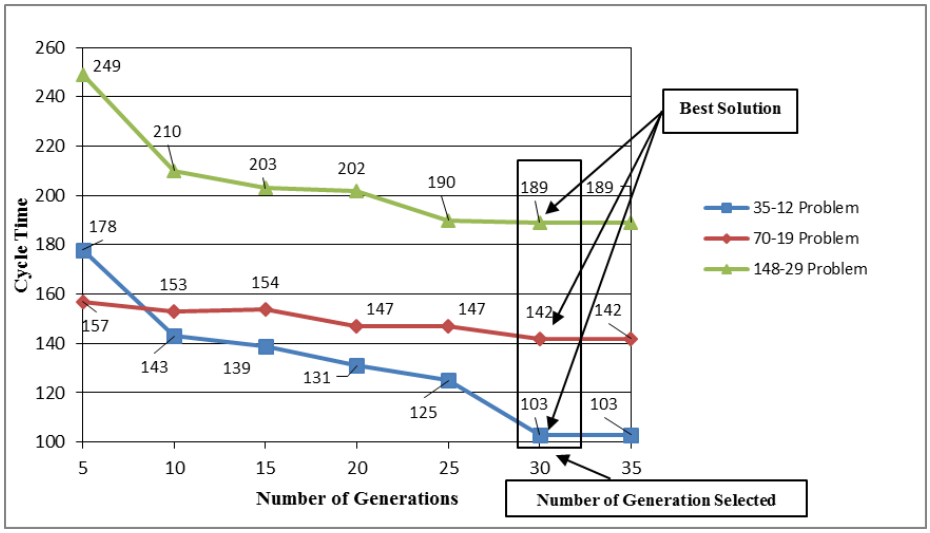

Figure 5.24 Performance of hybrid CS-PSO in terms of stopping condition

5.3.5 Computational study on PSO variants and Hybrid PSO models

All the four variants and hybrid models of PSO are tested on the 32 benchmark problems. Detailed results obtained using four variants and Hybrid PSO models are presented in this section. Table 5.19 reports the detailed comparison of the four variants (Set I) for the RALB problems where the variants are developed based on variations in the velocity update equation and Table 5.20 reports the detailed comparison of the hybrid models of PSO (Set II) of RALB problems which are developed by hybridizing PSO with two other metaheuristic algorithms for the objective of minimizing cycle time in a robotic assembly line.

The results obtained by evaluating 32 test problems are presented in Table 5.19 and Table 5.20. Column I shows the problem number. Column II shows the task size of the problems evaluated and Column III shows the number of workstations/robots for each problem. Column IV shows the cycle time for the 32 test problems, Column V

shows the modified cycle time for hGA algorithm. Results obtained from four variants of PSO are presented in the remaining columns. Computational time and percentage deviation in the results obtained are also presented. The first five columns are same for Table 5.20 and this table presents the details of the results obtained for the hybridized versions of PSO.

As reported earlier in Section 5.3.3, proposed RALB-2 is solved using the optimization solver Cplex. The results obtained with PSO variants and hybrid PSO models are compared with the Cplex solution. It is found that the results of all PSO variants and hybrid models are very close to Cplex solutions and it could be clearly observed that hybrid CS-PSO produces better results when compared to other variants. Detailed analysis on the variation in the objective function value (cycle time) when compared with the benchmark result reported by (Gao et al., 2009) is presented in the tables. It could be observed that all proposed PSO algorithms attained the same solution for the first three problems. Among all the proposed PSO algorithms, hybrid CS-PSO performs better in terms of quality of the solution. Average percentage deviation of the results obtained using PSO-W when compared with the benchmark data is found to be 3.0% for the thirty two problems, for PSO-C the average percentage deviation is higher when compared to PSO-W and it is calculated as 5.7%. Average percentage deviation for PSO-TVIW and PSO-TVAC is almost same and it is found to be 6.4%. For the hybridized PSO variants, the average deviation is found to be higher when compared with the other four variants. Average percentage deviation found for HPSO-W is 6.5%, for HPSO-C average percentage deviation in the objective function evaluated is 7.5% and for hybrid CS-PSO it is found to be 7.6%. So it can be concluded that hybridizing algorithms helps in improving the quality of the solution compared to other variants. The good performance of proposed hybrid CS-PSO is due to the implementation of local search schemes and different methods adopted to create new solution in the search space.

Quality of the solution is given importance compared to the computational time in this thesis. Table 5.18 and 5.19 also shows the computational time taken by all the proposed PSO algorithms. The proposed PSO variants to solve RALB-2 provide good quality solution in practical time. And it is observed that HPSO-C takes more time when compared to other variants. Among the hybridized variants the computational time for hybrid CS-PSO is lesser than HPSO-C and not much of difference when compared with

HPSO-W. HPSO-C variant needs to spend more time on selection of good quality solutions, local search and crossover operation. Results show that CPU times of the proposed hybrid PSO models are comparable to hGA for the problems up to 111 task problems.

In order to demonstrate the performance of the proposed PSO approaches, a real world problem from an automobile industry is chosen. The final assembly of an engine cradle is being undertaken (Gunther et al., 1983). Assembly consists of 35 tasks with 5 workstations and 5 robots to perform these tasks. Precedence relations and processing times of 35 tasks by 5 robots are presented in Appendix 1. All the proposed approaches of PSO are compared with the upper bound of hybrid GA. Table 5.18 shows the station and robot assignment with their cycle time. Amongst the proposed approaches it is analyzed that hybrid CS-PSO algorithm produces better results for RALB when compared to other proposed approaches. Cycle time obtained for hybrid CS-PSO is 332 and all the cycle time obtained by other proposed PSO algorithms are better than the result obtained using hybrid GA of 344.

5.3.6 Summary of the findings on RALB-2 problem study

In this section, a study on robotic assembly line balancing (RALB) problem with an objective of minimizing cycle time is considered. Optimization of cycle time is an important problem in a manufacturing industry. RALB problems falls under the category of NP-hard, hence an efficient metaheuristic algorithm (PSO) and its variants are proposed to solve the problem. Two heuristics (recursive and consecutive) are used to optimally allocate tasks to the workstations and assign the best fit robot to perform the tasks allocated. The heuristic is used to evaluate the cycle time of the balanced assembly line. Two heuristics are solved on benchmark datasets using standard PSO and the performance the proposed algorithms are compared with the benchmark results reported in the literature. Local exchange procedure is incorporated to improve the quality of the solution obtained through these methods and to escape from the local optima. Simulation experiments have been carried out over the thirty two bench mark data sets available in the literature. IBM Cplex Optimization studio Version 12.6.0.0 is also used to solve the problems. Cplex could only solve the first fourteen problems from the benchmark datasets. From the computational study conducted, it is concluded that consecutive allocation procedure performs better in terms of quality of the solution.

Hence, consecutive allocation procedure is solved using variants of PSO and hybrid PSO models. It is found that the results of PSO variants and hybrid PSO models are very close to Cplex solutions which show the effectiveness of the proposed algorithms. Computational results show that hybrid CS-PSO performs better among all the other proposed approaches when compared with benchmark results. However, computational time for hybrid CS-PSO variant is high for large size datasets but quality of solution is given importance compared to the computational time.

Table 5.18 Solutions Obtained for RALB-2 problem for 35 tasks by 5 robots

Hybrid GA Sequence (Gao et. al 2009) 1 5 10 6 7 8 14 12 17 18 19 9 20 15 16 21 22 23 24 13 25 26 27 2 3 4 30 34 31 32 11 28 33 35 29	Station Start points: 1 14 20 25 30 Robot Assignment: 4 3 1 5 2 Cycle Time: 344
Standard PSO Sequence-Recursive: 1 5 17 10 12 2 3 6 7 14 15 16 18 19 20 21 25 22 26 30 31 8 9 32 23 24 27 34 13 4 11 28 33 35 29	Station Split points: 1 3 19 31 34 Robot Assignment: 2 2 4 1 2 Cycle Time: 338
Standard PSO Sequence-Consecutive: 1 10 12 17 2 3 4 5 6 7 18 19 14 15 16 20 21 25 26 22 23 8 9 13 24 27 34 11 30 31 32 33 28 29 35	Station Start points: 1 4 14 22 34 Robot Assignment: 2 2 4 1 2 Cycle Time:336
PSO-W Sequence: 1 10 2 17 12 3 5 6 7 18 19 20 14 15 8 9 13 16 21 22 30 31 23 25 24 26 27 34 32 4 11 28 33 29 35	Station Start points: 1 6 15 31 34 Robot Assignment: 2 2 1 1 2 Cycle Time: 343
PSO-C Sequence: 1 17 10 12 5 6 7 2 14 15 16 18 19 3 4 20 21 22 23 30 24 31 25 26 27 8 9 13 32 11 28 33 35 34 29	Station Start points: 1 7 3 24 9 Robot Assignment: 2 2 4 1 2 Cycle Time: 341
PSO-TVIW Sequence : 1 2 17 12 10 5 6 7 18 19 14 15 16 20 21 25 26 22 23 24 27 30 8 9 13 31 32 34 3 4 11 28 33 35 29	Station Start points: 1 6 16 24 34 Robot Assignment: 2 2 4 1 2 Cycle Time: 341
PSO-TVAC Sequence: 1 10 12 2 17 5 6 7 14 15 18 19 20 8 16 21 30 22 23 25 26 3 4 24 27 9 3 1 32 13 34 11 28 33 35 29	Station Start points: 1 6 20 25 31 Robot Assignment: 2 2 1 4 2 Cycle Time: 338
HPSO-W Sequence: 1 2 17 10 5 12 3 4 6 7 18 19 20 14 15 16 21 25 26 22 8 23 24 30 31 27 34 9 13 32 11 33 28 29 35	Station Start points: 1 3 20 22 34 Robot Assignment: 2 2 4 1 2 Cycle Time: 336
HPSO-C Sequence: 1 17 5 6 7 10 14 15 16 12 18 19 20 2 3 4 21 30 25 22 23 24 26 27 8 9 13 11 31 32 33 34 28 35 29	Station Start points: 1 14 2 22 13 Robot Assignment: 4 2 2 1 2 Cycle Time: 334
Hybrid CS-PSO Sequence: 110 17 5 6 8 2 3 7 12 18 14 15 16 19 20 21 25 26 22 23 4 9 24 27 30 31 32 13 34 11 33 28 29 35	Station Start points: 1 2 15 22 31 Robot Assignment: 4 2 4 1 2 **Cycle Time: 332**

Table 5.19 Results obtained by PSO variants for RALB-2 problems- Set I

Problem Number	Datasets	hGA (Gao et al., 2009)		PSO-W			PSO-C			PSO-TVIW			PSO-TVAC		
		Cycle Time	Modified CPU Time (sec)	Cycle Time	CPU Time (sec)	% age deviation	Cycle Time	CPU Time (sec)	% age deviation	Cycle Time	CPU Time (sec)	% age deviation	Cycle Time	CPU Time (sec)	% age deviation
1	25-3	503	2	503	2.3	0	503	2.9	0	503	3.2	0	503	3.2	0
2	25-4	327	2	327	2.4	0	327	3	0	327	3	0	327	3.1	0
3	25-6	213	2	200	2.7	6.1	200	3.5	6.1	200	3.8	6.1	200	4.5	6.1
4	25-9	123	3	115	3	6.5	114	3.8	7.31	110	4.3	10.56	110	5.1	10.6
5	35-4	449	5	376	3.7	16.26	342	3.9	23.83	341	3.9	24.05	342	4.1	23.83
6	35-5	344	7	343	4	0.29	341	4.5	0.87	341	5.2	0.87	338	6.8	1.74
7	35-7	222	9	216	7	2.7	211	7.2	4.95	211	7.2	4.95	212	7.4	4.5
8	35-12	113	12	104	6.2	7.96	103	7.5	8.84	103	7.9	8.84	103	8.7	8.84
9	53-5	554	7	512	13.3	7.58	453	13.8	18.23	454	14	18.05	453	14.2	18.23
10	53-7	320	9	324	14.5	-1.25	294	15.2	8.125	293	15.8	8.43	295	16.3	7.81
11	53-10	230	11	224	16.8	2.61	227	18.3	1.3	231	18.5	-0.43	234	18.6	-1.73
12	53-14	162	14	146	18.4	9.88	144	19.2	11.1	142	19.2	12.34	145	19.8	10.49
13	70-7	449	16	447	28.5	0.45	444	30.2	1.11	429	30.5	4.45	429	29.5	4.45
14	70-10	272	22	261	28.8	4.04	259	30.6	4.77	258	30.8	5.14	263	29.8	3.3
15	70-14	204	26	199	36	2.45	197	37.2	3.43	195	37.6	4.41	195	37.3	4.41
16	70-19	154	33	144	50.2	6.49	142	51.5	7.79	146	51.8	5.19	143	52	7.14
17	89-8	494	19	461	50.9	6.68	463	51.8	6.27	459	52.2	7.08	468	52.2	5.26
18	89-12	370	24	354	52.1	4.32	315	53.5	14.86	314	54.2	15.13	311	55.1	15.94
19	89-16	236	29	247	56.6	-4.66	231	58.2	2.11	221	58.5	6.35	233	60.2	1.27
20	89-21	205	36	171	62.2	16.59	177	64.2	13.65	177	65.1	13.65	173	73.2	15.6
21	111-9	557	64	529	87.4	5.03	528	90.2	5.2	535	91.8	3.94	528	92.6	5.2
22	111-13	319	78	321	93.1	-0.63	347	94.8	-8.77	322	97.5	-0.94	322	100.9	-0.94
23	111-17	257	93	246	93.6	4.28	250	96.6	2.72	253	96.8	1.55	249	98.9	3.11
24	111-22	192	110	201	98.2	-4.69	203	108.5	-5.72	203	110.2	-5.72	189	110.2	1.56
25	148-10	600	98	628	195.2	-4.67	585	200.5	2.5	603	200.8	-0.5	582	201.2	3
26	148-14	427	121	412	206.1	3.51	417	206.5	2.34	421	207.2	1.405	431	207.6	-0.93
27	148-21	300	135	292	205.3	2.67	272	208.6	9.33	273	210.8	9	284	218.5	5.33
28	148-29	202	169	203	215.8	-0.5	187	220.8	7.42	189	222.3	6.43	189	225.5	6.43
29	297-19	646	335	692	1123.5	-7.12	676	1358.2	-4.64	646	1380.4	0	635	1413.2	1.7
30	297-29	430	369	428	1135.9	0.47	399	1438.6	7.2	392	1438.6	8.83	400	1456.2	6.97
31	297-38	344	405	328	1140.2	4.65	294	1440.4	14.53	294	1440.4	14.53	293	1450.6	14.82
32	297-50	256	448	256	1148.7	0	234	1458.8	8.59	225	1453.8	12.1	225	1458.8	12.1
				Avg % age Dev.		**3.1%**	**Avg % age Dev.**		**5.8%**	**Avg % age Dev.**		**6.4%**	**Avg % age Dev.**		**6.4%**

Table 5.20 Results obtained by hybrid PSO models for RALB-2 problems-Set II

Problem Number	Datasets	hGA (Gao et al., 2009)		HPSO-W			HPSO-C			Hybrid CS-PSO		
		Cycle Time	Modified CPU Time (sec)	Cycle Time	CPU Time (sec)	% age deviation	Cycle Time	CPU Time (sec)	% age deviation	Cycle Time	CPU Time (sec)	% age deviation
1	25-3	503	2	503	3.4	0	503	3.6	0	503	3.6	0
2	25-4	327	2	327	3.3	0	327	3.5	0	327	3.9	0
3	25-6	213	2	200	4.2	6.1	200	4.8	6.1	200	4.2	6.1
4	25-9	123	3	112	4.9	8.94	110	5.2	10.56	110	4.5	10.56
5	35-4	449	5	342	4.3	23.83	342	4.5	23.83	341	5.2	24.05
6	35-5	344	7	336	6.8	2.32	334	7.1	2.9	332	6.3	3.48
7	35-7	222	9	214	7	3.6	211	7.2	4.95	211	6.9	4.95
8	35-12	113	12	103	8.5	8.84	103	8.9	8.84	103	8.9	8.84
9	53-5	554	7	452	13.6	18.41	450	13.6	18.77	449	13.5	18.95
10	53-7	320	9	295	15.1	7.81	294	15.1	8.12	294	16.8	8.12
11	53-10	230	11	234	17.5	-1.73	234	17.6	-1.73	221	17.9	3.91
12	53-14	162	14	144	19.4	11.11	143	19.6	11.72	142	20	12.34
13	70-7	449	16	430	29.8	4.23	430	30.8	4.23	430	32.9	4.23
14	70-10	272	22	262	29.6	3.67	256	31.2	5.88	264	35.8	2.94
15	70-14	204	26	195	37.4	4.41	194	38.7	4.9	194	43.3	4.9
16	70-19	154	33	141	51.6	8.44	140	51.9	9.09	140	47.8	9.09
17	89-8	494	19	460	54.5	6.88	458	56.1	7.28	460	45.7	6.88
18	89-12	370	24	314	56.2	15.13	308	58.8	16.75	320	51.6	13.51
19	89-16	236	29	223	62.3	5.5	222	65.3	5.93	219	63.3	7.2
20	89-21	205	36	172	82.3	16.09	170	85.2	17.07	170	80.5	17.07
21	111-9	557	64	524	93.2	5.92	524	94.4	5.92	523	85.5	6.1
22	111-13	319	78	322	98.2	-0.94	322	98.4	-0.94	321	92.5	-0.62
23	111-17	257	93	247	99.2	3.89	244	104.4	5.05	240	107.4	6.61
24	111-22	192	110	184	108.8	4.16	182	110.5	5.2	182	114.5	5.2
25	148-10	600	98	585	201.5	2.5	580	202.5	3.33	593	183.5	1.16
26	148-14	427	121	421	207.8	1.4	416	208.5	2.57	419	207.9	1.87
27	148-21	300	135	285	220.9	5	274	222.4	8.66	273	219.5	9
28	148-29	202	169	188	228.1	6.93	187	229.1	7.42	189	242.2	6.43
29	297-19	646	335	606	1423.7	6.19	597	1435.5	7.58	594	1118.3	8.04
30	297-29	430	369	411	1460.2	4.41	396	1468.2	7.9	394	1331.3	8.37
31	297-38	344	405	303	1465.8	11.91	295	1475.6	14.24	305	1593.5	11.33
32	297-50	256	448	244	1472.9	4.68	232	1488.2	9.37	221	1664.3	13.67
				Avg % age Dev.		**6.5%**	**Avg % age Dev.**		**7.5%**	**Avg % age Dev.**		**7.6%**

5.4 U- Shaped RALB problem

Assembly lines are classified mainly into two types based on the nature of flow: straight (traditional) assembly lines with single and multi/mixed products and U-shaped assembly lines (also called U-lines) with single and multi/mixed products. Implementation of U-shaped layout helps the manufacturer to increase or decrease the number of operators based on the change in demand (Aigbedo and Monden, 1997). Distinguishing feature of U-shaped assembly lines is that it allows task to be assigned to the workstations after all its predecessor or successor are assigned to the earlier or same workstation. This feature of U-shaped assembly line balancing problems allows for the forward and backward assignment of tasks to workstations (Kara, 2008). U-shaped layout helps to reduce the cycle time and cost of assembly which attracts different industries to implement this system (Kubota, 2011, Yalaoui et al., 2013).

U-shaped layout was introduced at a subsidiary factory of Toyota Motor Corp. Implementation helped the company to save the capital expenditure and reduce the production time. By implementing U-shaped layout more than one task can be performed at once on a vehicle, such as installing the engine in the front while adding underbody parts in the back. In automobile assembly, vehicles are carried on a conveyor belt and the length of the assembly line could be reduced to one-third of the length when compared to traditional assembly lines. The company could save up to 40 percent in their total capital expenditure by implementing U-shaped assembly line. (Kubota, 2011).

The largest integrated moving assembly line the world was introduced by Boeing in 2010 after implementing U-shaped assembly line in their manufacturing industry (Fetters-Walp, 2010). New record rate of assembling 8.3 aircraft per month or 100 per year could be achieved by Boeing after implementing U-shaped assembly layout. The company could reduce the flow time and production cost. By implementing U-shaped assembly lines an easier work environment could be created for the operators (Boeing, 2014).

Robotic assembly lines are widely used in these industries. RALB problem is the generalized form of tradition assembly line balancing problem. Few researchers have proposed algorithms to solve the problem of this nature. However, there is no research undertaken in the field of robotic assembly line-balancing problem with a U-shaped

configuration (RUALB). This thesis mainly aims to propose efficient metaheuristic based search algorithm to solve U-shaped robotic assembly line balancing (RUALB) problem. The main objective of the algorithm is to minimize the cycle time when the number of workstations are fixed in a U-shaped assembly line layout. Following sections present the details of U-shaped robotic assembly line balancing and algorithms used to solve the problem.

5.4.1 PSO to solve RUALB problem

It is already proven by many researchers that simple assembly line balancing problem is NP-hard (Gutjahr and Nemhauser, 1964). RUALB problem is further complicated with addition of robots and U-shaped configuration. Hence, RUALB is also NP-hard. Different metaheuristics are used to solve problems of this nature. In this research, PSO algorithm which has achieved great success in optimizing engineering problems (Guo et al., 2013)are used to solve RUALB problem. As explained in the previous section, PSO is a population-based stochastic optimization technique

PSO is developed based on the social behavior of organisms. Basic advantages of using particle swarm optimization algorithm to solve this problem (Lee and Park, 2006, Hu et al., 2014) are: simple to implement when compared to other evolutionary algorithms, there are no overlapping and mutation calculation and very less parameters to fine tune.

The pragmatic issues in using or any other metaheuristics algorithms are to find optimal control parameters. PSO parameters optimized in this thesis are initial population, acceleration coefficients and stopping condition. The experimental studies followed to obtain the optimal parameters are explained in Section 5.5.5.

5.4.2 Initial Population and Initial velocity

PSO algorithms start with an initial population and initial velocity. The same procedure adopted in Section 5.1.1 to solve straight robotic assembly line is used for solving RUALB problem. However the fitness evaluation procedure is different from the one presented for RALB. A new fitness evaluation method is developed for solving RUALB problem with an objective of minimizing cycle time in a U-shaped robotic assembly line. Section 5.4.3 presents the details on how the cycle time (objective function) is evaluated.

5.4.3 Fitness Evaluation in RUALB (Task and robot allocation)

A consecutive heuristic procedure proposed by Levitin et al. (2006) for straight robotic assembly line is adopted in this thesis. This heuristic allocates tasks and robots to workstations with an objective of minimizing the cycle time of the assembly line. An efficient method is developed for allocating tasks and robots to workstations in a U-shaped robotic assembly line.

Procedure starts with an initial cycle time of the assembly line, C_0. Major objective of the procedure is to allocate maximum number of tasks at each workstation. Tasks are allocated to the workstation in U-shaped assembly line by moving forward and backward through the precedence diagram in contrast to a typical forward move in the traditional assembly systems. Best robots to perform the tasks allotted to the workstations are checked for allotment. C_0 is incremented by 'one' if certain tasks cannot be assigned within the given initial C_0 value and this procedure continues until all the tasks are assigned to all the workstations. This section presents the stepwise procedure adopted for finding out the cycle time of the U-shaped robotic assembly line.

Step 1: C_0, the initial value of cycle time is the mean of the minimum performance time of robots for the tasks.

Initial assembly line time $C_0 = [\sum_{j=1}^{N_a} \min_{1 \le i \le N_r} t_{ih} / N_w]$ **(5.29)**

The following feasible sequence of tasks (Figure 5.25) is considered for illustration. Initial C_0 is calculated using the robot performance times as shown in Table 5.2.

1	3	2	4	5	6	7	9	8	10	11

Figure 5.25 Example sequence considered for illustration

The sequence meets the precedence constraints. Initial C_0 for the example is found out to be 109 where 37,42,38,40,25,65,50,34,33,41,38 are the minimum robot task times (refer Table 5.2).

C_0= (37+42+38+40+25+65+40+34+33+41+38)/4=108.25

Step 2: First workstation is opened and tasks are allocated in such a way the procedure chooses the tasks from either side of the sequence and checks if one or more robot can

perform the allocated tasks within C_0. The set of assignable tasks are determined by all those tasks whose predecessors or successors have already been assigned. Tasks are allocated to workstations by moving forward and backward through the precedence diagram.

Each workstation *'s'* has a set of preferred/ allotted robots *H* which is defined as follows:

$$r \in H, if \;\; m(r) \geq m(h), \;\; \text{for } 1 \leq h \leq N_r \qquad \textbf{(5.30)}$$

Here *m (h)* is the maximal number of tasks a robot *h* can perform in the given sequence *sq* during a time lesser than C_0.

$$T_s(h) = \sum_{r=p1_S}^{p1_S+m(h)} t_{h,sq(r)} < C_0 \leq \sum_{r=p1_S}^{p1_S+m(h)+1} t_{h,sq}(r) \qquad \textbf{(5.31)}$$

Next, it defines the robot to be assigned to the workstation *s* as:

$$h(s) = r, \;\; if \;\; T_s(r) < T_s(h) \;\; \forall h \in H \qquad \textbf{(5.32)}$$

Step 3: The start position of the next stations ($p1_{s+1}$) is calculated,

$$p1_{s+1} = pr_s + 1 = p1_s + m(\,h\,(s)) + 1 \qquad \textbf{(5.33)}$$

Repeat Step 2 and 3, until all tasks are assigned to given number of workstations**.**

Step 4: By repeating Steps 2 and 3, if there are still some more tasks left to be assigned to the workstations, C_0 is incremented *by 'one'* and steps 2 to 3 are repeated until all tasks are allotted to the given number of workstations.

Step 5: Best fit robot is assigned to each workstation. Best fit robot is determined based on the objective of minimizing cycle time. Table 5.21 shows how the best fit robot is selected for the example shown in Figure 5.27. Three tasks (1, 11, and 10) are assigned in the workstation 1 and Robot 2 is assigned to this workstation as explained in Table 5.21. Robot 2 performs the assigned tasks in lower time compared to other robots. Hence, Robot 2 is the best fit robot for Workstation 1. The total robot task time for each robot is calculated using Table 5.2.

Table 5.21 Illustration of best fit robot selection

Workstation	Tasks Assigned	Total Robot task time
1	1, 10 and 11	Robot 1: 81+45+76=202 **Robot 2: 37+46+38=121** Robot 3: 51+41+83=175 Robot 4: 49+77+87=213

Step 6: The maximum workstation time (sum of the minimum robot performance time for the tasks assigned to the station) is the cycle time for the sequence.

For the sequence shown in Figure 5.25, C_0 is calculated and is found to be 109. Procedure tries to initially allocate the tasks to the workstation and assign the best fit robots within the initial C_0 but the initial cycle time cannot allocate tasks 7 and 9 as shown in Figure 5.26. To accommodate all the tasks C_0 is incremented by one and when C_0 reaches 121 all tasks could be assigned and robot is assigned to perform all tasks as shown in Figure 5.27. Shaded portion in Figure 5.27 shows the tasks and robot allocation details.

5.4.4 Differences between straight and U-shaped robotic assembly line

In a straight line configuration workstations are positioned on a straight line whereas in case of U- shaped assembly line, the line is configured into a U-shaped configuration. Figure 5.28 shows a sample solution where tasks are assigned in a straight line for the sequence mentioned in the previous Section 5.1.1. Figure 5.29 shows a sample solution for tasks assigned in U-shaped form.

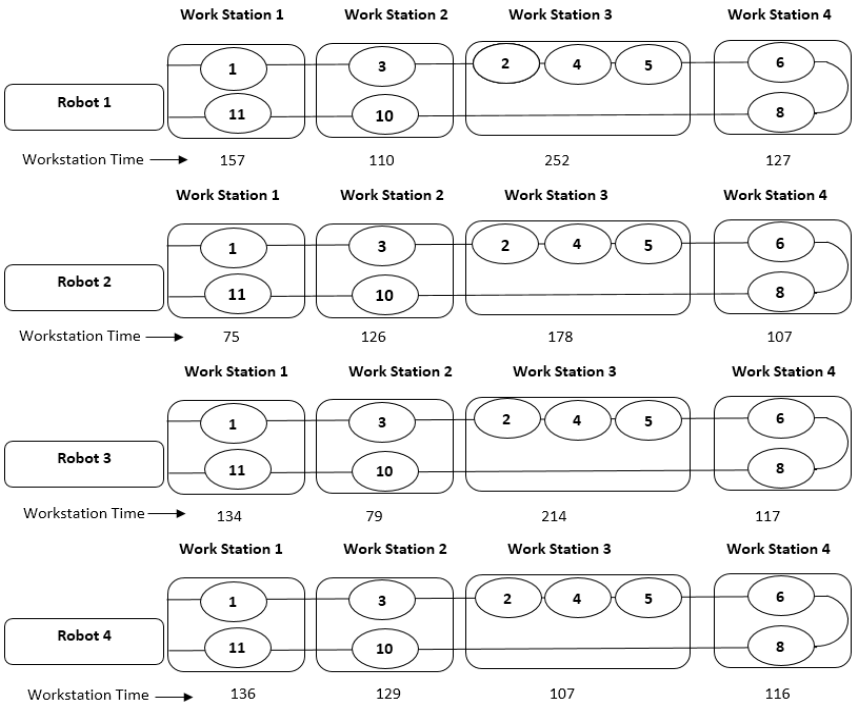

Figure 5.26 Example of assignment procedure for initial cycle time

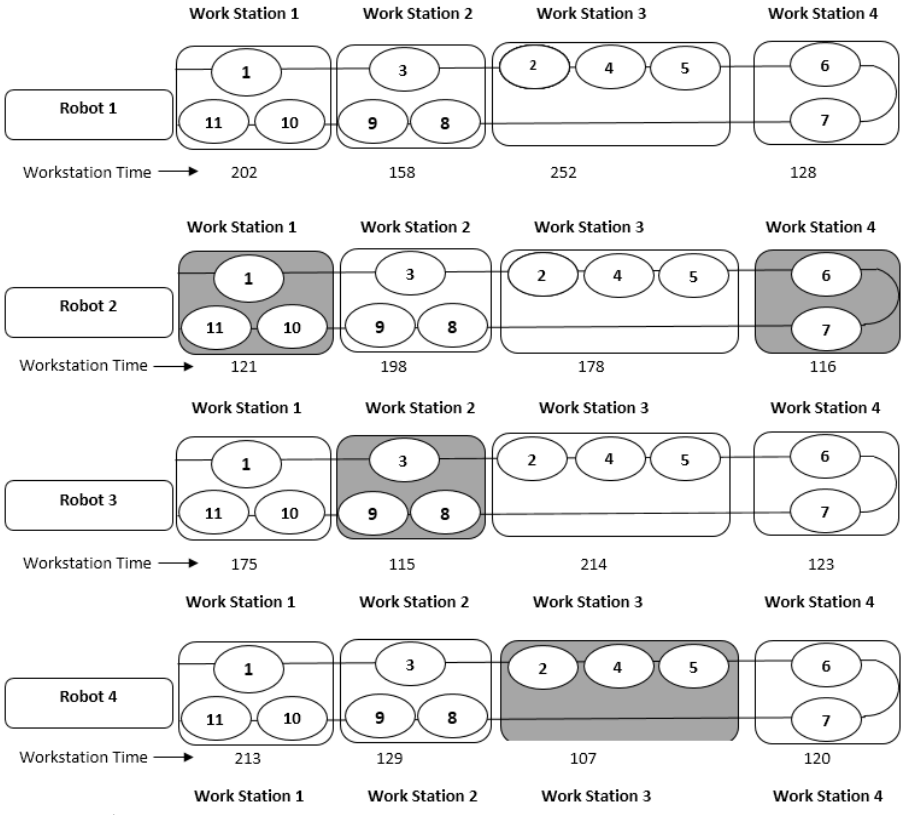

Figure 5.27 Final Assignment solution for U-shaped RALB

***(numbers within oval shapes represent the task number)*

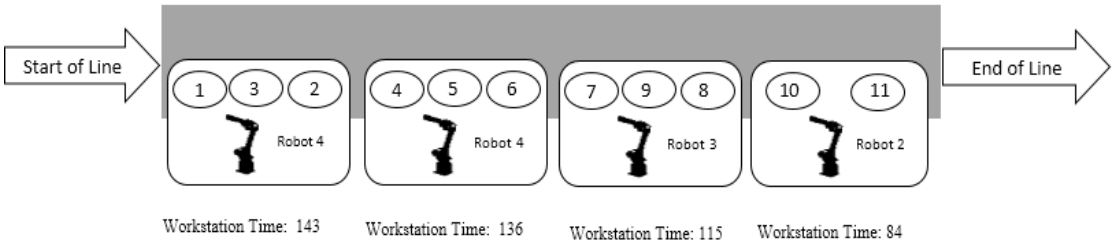

Figure 5.28 Straight Robotic Assembly Line

From Figure 5.28, it is understood that tasks are assigned in the order of the sequence generated without violating the precedence constraints for straight robotic assembly line. For U-shaped assembly line, tasks are allocated by searching forward and backward through the precedence diagram (numbers within oval shapes represent the task number). In case of U-shaped robotic assembly line, each task and any of its successor and/or predecessor can be allocated in the same workstation. U-shaped layout is easier to relocate the robot to balance the work load depending on the demand. This flexibility and adaptability of U-shaped layout makes it more attractive approach compared to straight line. The cycle time for the given sample is found to be 143 for straight robotic assembly line and for U-shaped robotic assembly line the cycle time is found to be 121 (Figure 5.29).

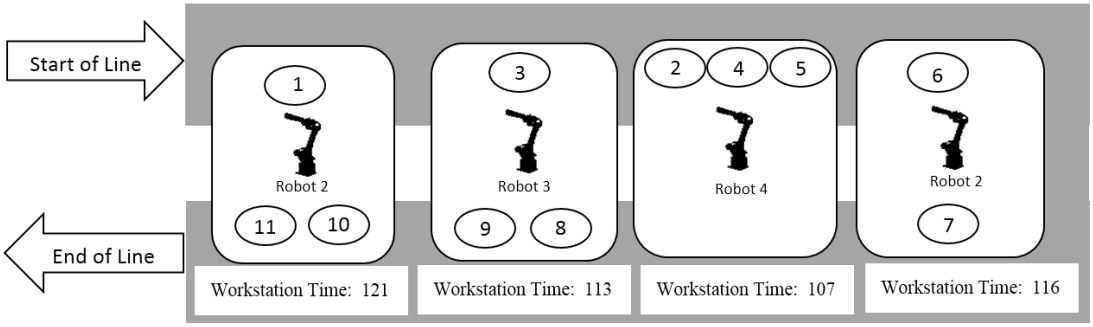

Figure 5.29 U-shaped Robotic Assembly Line

5.4.5 Computational Study for RUALB problems

To test the performance of the proposed PSO on RUALB problems computational experiments are conducted. The proposed algorithm is coded in C++ and the performances are tested on Intel core i5 processor (2.3 GHz). The details of the experiments conducted are presented in this section.

5.4.5.1 Datasets for computational study

The datasets used for computational study for straight robotic assembly line problems is used for evaluating the performance of RUALB problems also. Details of the precedence graphs used are available in http://www.assembly-line-balancing.de/. Thirty two test problems generated by Gao et al. (2009) for robotic assembly line are used in this research.

5.4.5.2 Parameters used for PSO to solve RUALB problems

Parameters selected influence the performance of PSO. The parameters used to solve RUALB problems are chosen based on the experiments conducted in order to get a satisfactory solution quality in an acceptable computational time. Experiments are performed to test the influence of each parameter on the solution quality. To find the best combination of parameters, three datasets of different size are chosen. Parameters with different combination are tested until the best combination is achieved. Quality of the solution is given utmost importance and not on the computational time in selecting the best set of parameters, Following are the parameters tested and used in the proposed PSO to solve RUALB problems.

Stopping condition: Proposed algorithm is terminated if the iteration approaches a predefined maximum number of generations or sufficiently good fitness value. In this case a predefined maximum number of iterations (generations) is used. Different stopping conditions are tested such as 5,10,15,25 and 30. Three problems of different sizes based on the task size are tested for these different stopping condition and it is observed that, the best solution could be attained when the number of iterations (generations) is set to 25. Figure 5.30 illustrates the performance of the PSO algorithm based on the stopping condition for three selected datasets (small, medium and large size dataset). Problems are 35-12, 70-19, 148-29 where 35, 70, 148 are the number of tasks whereas 12, 19, 29 indicates the number of robots and work stations available to perform the tasks. The numbers in the graph represent the cycle time obtained for the three selected problems under different stopping conditions. And after 25^{th} iteration, the algorithm started converging to same solution. The stopping condition (number of generations) for all the problems tested for RUALB problems is set to 25.

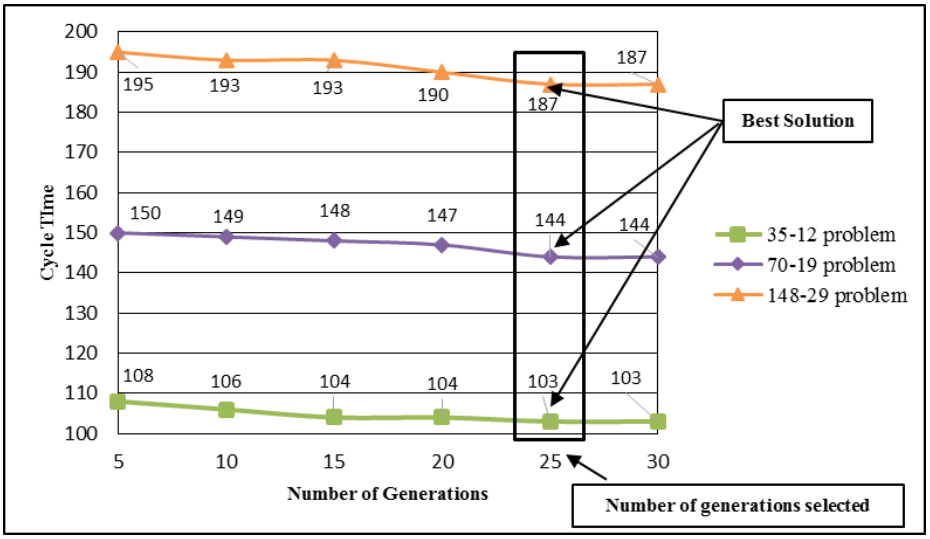

Figure 5.30 Performance of PSO in terms of stopping condition

Acceleration coefficients: The same set of different combination of acceleration coefficients used to solve PSO in straight robotic assembly line is also tested for solving RUALB problems**.** Table 5.12 shows the different combinations of c_1, c_2 and c_3 tested. Three problems are taken into consideration for the testing. Problems selected are 53-5, 297-19, and 148-14 where 53,148,297 refer to the number of tasks and 5, 19 and 14 indicates the number of robots and work stations available to perform the tasks. From the Figure 5.31 it is analyzed that Group D (c_1=1, c_2=2 and c_3=2) produced best solutions and this combination of acceleration coefficients are used to solve all the problems.

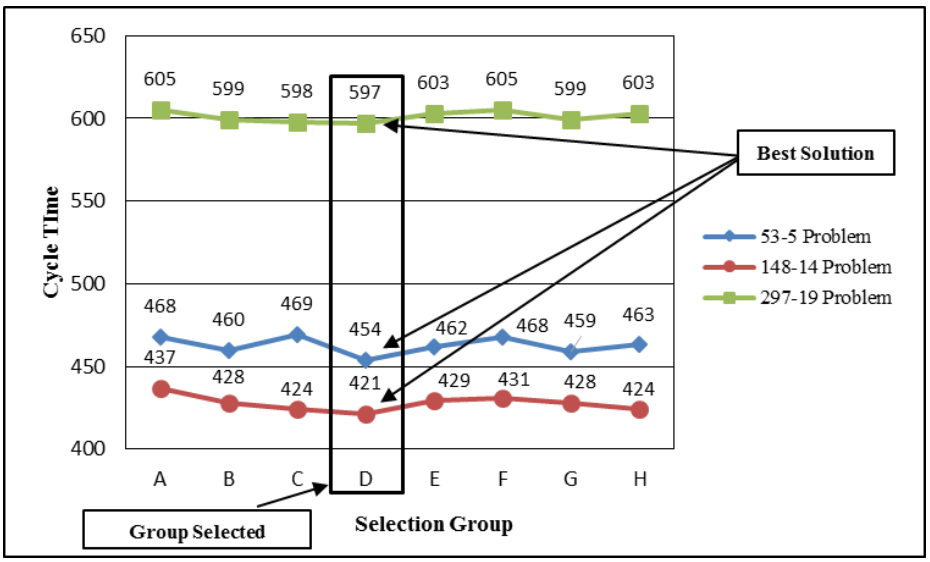

Figure 5.31 Selection of acceleration coefficients based on the performance of the algorithm

5.4.5.3 Performance of u-shaped robotic assembly line versus straight RALB

32 test problems available from the literature are evaluated using the proposed PSO algorithm. Due to non-deterministic nature of the problem and algorithm, it is necessary to run same problem multiple times to obtain the optimal solution. All problems are run ten times and most of the runs converged to same solution for most of the problems. The results obtained by evaluating 32 test problems for RUALB are presented in Table 5.22. Column I shows the type of problem. The problems are classified into three categories: small (up to 35 tasks), medium (up to 89 tasks) and large (above 100 tasks). Column II in table shows the problem name (source of the file). Table 5.13 presents the source of the datasets evaluated. Column III and Column IV shows the number of tasks and number of workstations considered for the evaluation. Column V reports the cycle time for straight robotic assembly line as reported by Gao et al. (2009). Column VI shows the results obtained from Cplex (optimization software). Cplex can only find optimal solutions for four small size problems (25-3, 25-4, 25-6 and 25-9) in an acceptable computational time. Cplex could not get the solutions for the rest of the problems. Column VII reports the cycle time obtained for the same problems for the U-shaped robotic assembly line using the proposed PSO. Column VIII shows the percentage deviation of the cycle time and Column IX reports the average percentage deviation for the three problem categories.

From the results presented in Table 5.22 it can be observed that proposed PSO algorithm finds better solution for 28 out of 32 instances. When comparing the solutions obtained for U-shaped robotic assembly line with straight robotic assembly line, the proposed PSO algorithm generate better results for all small size problems. Average improvement in the cycle time for small size problems is found to be 6.99%. Average improvement in the cycle time for medium size problems is found to be 6.32% and it is observed that out of 12 problems only 2 problems could not yield better solution when compared to straight robotic assembly line. Average percentage improvement for large size category is found to be 4.85%. It may be noted that 10 out of 12 problems (large size) yield better solution when compared with straight robotic assembly line. From the results in Table 5.22 it is clearly evident that the cycle time of U-shaped robotic assembly line is better than that of straight robotic assembly line. For the problem addressed here, the selection of best available robots helps to reduce the cycle time and in turn increases the productivity of the assembly line.

Simple assembly line balancing problems falls under NP-hard category (Gutjahr and Nemhauser, 1964). RUALB problem is further complicated with addition of robots and U-shaped configuration. Hence, RUALB is also NP-hard. Computational time required to reach a solution is required for calculating the complexity of the problem addressed. Section 5.3.3 gives the details of complexity measures used for RALB problems same set of complexity measures are used for RUALB problems. Details of the complexity measures used for RUALB problem.

1. **Assembly Flexibility**-Problems with eight levels of F-ratio are generated and evaluated. Figure 5.32 shows F-ratio versus Computational time for 8 problems sets for RUALB allocation procedure. Computational time is presented in Table 5.23. It is noted that computational time is an increasing function of F-ratio. A high F-ratio indicates that there are fewer alternatives for assigning tasks to workstations where as low F-ratio gives different ways of assignment**.**
2. **WEST Ratio**- WEST Ratio of the problems of RUALB is similar to the WEST Ratio of RALB. Table 5.14 gives the details of WEST Ratio of the problems evaluated.
3. **Relative Complexity**- Table 5.23 presents the relative complexity of the problem addressed. Relative complexity is calculated based on the computational time taken by each problem. The relative complexity of the problem increases with the increase in task size of the problem.

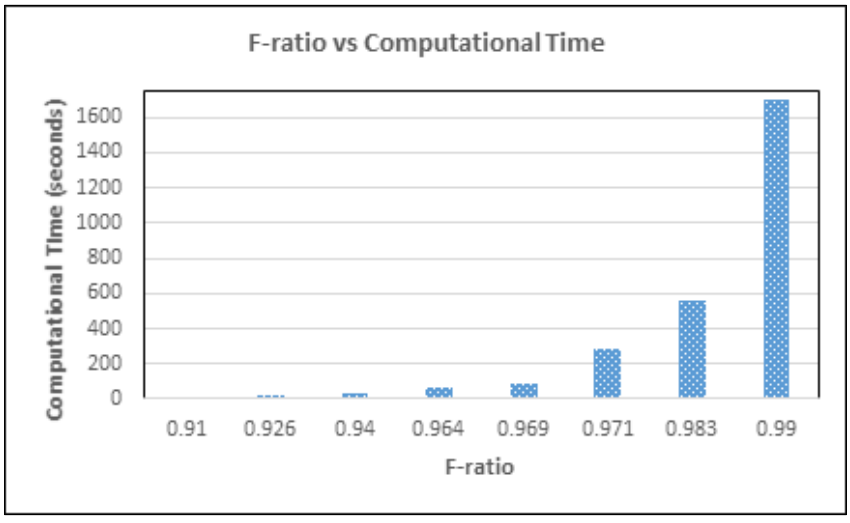

Figure 5.32 F-ratio vs Computational Time for RUALB problems

Table 5.22 Results of the 32 benchmark problems for RUALB

Problem Type	Problem Name	Tasks	Work Stations/Robots	Cycle Time			% deviation	Average % Deviation
				Straight RALB (Gao et al., 2009)	Cplex	PSO RUALB		
Small Size Problem	Rosenberg	25	3	503	500*	500	0.5	
			4	327	318*	318	2.75	
			6	213	188*	188	11.73	
			9	123	114*	114	7.31	
	Gunther	35	4	449	-	355	20.93	
			5	344	-	332	3.48	
			7	222	-	221	0.45	6.99%
			12	113	-	103	8.84	
Medium Size Problem	Hahn	53	5	554	-	459	17.14	
			7	320	-	286	10.62	
			10	230	-	220	4.34	
			14	162	-	148	8.64	
	Tonge	70	7	449	-	447	0.44	
			10	272	-	272	0	
			14	204	-	**211**	**-3.43**	
			19	154	-	144	6.49	
	Lutz	89	8	494	-	**496**	**-0.49**	
			12	370	-	326	11.89	
			16	236	-	224	5.08	
			21	205	-	174	15.12	6.32%
Large Size Problem	Arcus	111	9	557	-	545	2.15	
			13	319	-	**320**	**-0.313**	
			17	257	-	256	0.38	
			22	192	-	186	3.12	
	Bartholdi	148	10	600	-	**629**	**-4.83**	
			14	427	-	421	1.40	
			21	300	-	283	5.66	
			29	202	-	187	7.42	
	Scholl	297	19	646	-	597	7.58	
			29	430	-	394	8.37	
			38	344	-	293	14.8	
			50	256	-	224	12.5	4.85%

To test the performance of the proposed algorithm an actual assembly line balancing problem is used. The final assembly operation of a major automobile manufacturer where the assembly of engine cradles is considered (Gunther et al., 1983). In the problem there are 35 tasks to be performed and 5 robots are chosen to perform these tasks in the assembly line. The problem consists of 45 direct precedence relations

among 35 tasks and processing times of 35 tasks by 5 robots are shown in Appendix 1. The solutions obtained for the problem using PSO for U-shaped assembly line and solution obtained for straight robotic assembly line using hybrid GA are reported in Table 5.24. The table shows the details of the tasks and robots to be allocated at each workstation. In summary, cycle time for the U-shaped robotic assembly line is lower when compared to the straight robotic assembly line.

Table 5.23 Relative Complexity of RUALB problem

Problem No:	Problem Dataset	Computational Time	Relative Complexity	Problem No:	Problem Dataset	Computational Time	Relative Complexity
1	25-3	8.0	0	17	89-8	84.5	9.56
2	25-4	9.2	0.15	18	89-12	87.1	9.88
3	25-6	10.5	0.31	19	89-16	91.2	10.40
4	25-9	13.5	0.68	20	89-21	95.3	10.91
5	35-4	16.8	1.10	21	111-9	234.2	28.27
6	35-5	19.5	1.43	22	111-13	253.7	30.71
7	35-7	27.5	2.43	23	111-17	298.5	36.31
8	35-12	31.5	2.93	24	111-22	348.9	42.61
9	53-5	32.5	3.06	25	148-10	445.8	54.72
10	53-7	34.8	3.35	26	148-14	519.2	63.9
11	53-10	35.6	3.45	27	148-21	595.1	73.38
12	53-14	41.2	4.15	28	148-29	655.3	80.91
13	70-7	60.1	6.51	29	297-19	1573.2	195.65
14	70-10	66.8	7.35	30	297-29	1693.8	210.72
15	70-14	72.4	8.05	31	297-38	1752.9	218.11
16	70-19	82.2	9.27	32	297-50	1802.3	224.28

5.4.6 Summary of the findings on RUALB problem

Robotic U-shaped assembly line balancing (RUALB) problem is considered in the previous section. It is found from the literature that very few researchers have worked on this problem. To solve the problem of minimizing cycle time in a U-shaped robotic assembly line, particle swarm optimization algorithm is proposed. The tasks and robot assignment in U-shaped configuration is highly complex when compared to straight assembly line. The performance of the proposed PSO on RUALB reported in this section is for the benchmark problems with eight precedence graphs only. Thirty two benchmark problems originally proposed by earlier researchers to solve RALB, is used to the test the performance of RUALB. Different parametrical studies are conducted to do the computational analysis. Results shows that proposed PSO algorithm for robotic

U-shaped assembly line reports minimal cycle time when compared to that of straight robotic assembly line for twenty eight out of thirty two problems. From the computational study, results shows that computational time for the proposed algorithm is high large data sets which could be due to large search space in U-shaped layout setup. Different complexity measures are also presented.

Table 5.24 Solutions Obtained for 35 task problem with 5 robots

Hybrid GA Task Sequence for Straight RALB: 1 5 10 6 7 8 14 12 17 18 19 9 20 15 16 21 22 23 24 13 25 26 27 2 3 4 30 34 31 32 11 28 33 35 29			
Workstation Number	**Tasks Allocated**	**Robot**	**Workstation Time**
Workstation 1	1 5 10 6 7 8	Robot 4	332
Workstation 2	14 12 17 18 19 9	Robot 3	**344**
Workstation 3	20 15 16 21 22 23 24 13	Robot 1	344
Workstation 4	25 26 27 2 3 4	Robot 5	330
Workstation 5	30 34 31 32 11 28 33 35 29	Robot 2	333
PSO Task Sequence for RUALB: 1 12 10 5 6 7 18 14 19 15 2 17 16 20 21 22 30 8 23 3 25 31 4 9 11 13 24 26 27 32 28 33 34 35 29			
Workstation Number	**Tasks Allocated**	**Robot**	**Workstation Time**
Workstation 1	1 29 34 2 3 35 28 33	Robot 2	322
Workstation 2	4 27 26 32 31 30 24	Robot 1	311
Workstation 3	11 17 10 5 6 8 9	Robot 2	**333**
Workstation 4	7 23 25 22 21 20	Robot 4	331
Workstation 5	14 19 15 16 13 18 12	Robot 2	314

5.5 Summary

In this chapter, RALB problem with an objective of minimizing cycle time is considered. Optimizing cycle time is an important problem in manufacturing systems. Since the problems addressed here is well known as NP-hard, metaheuristic algorithms are developed to solve the problem. Two heuristics are used for the effective allocation of tasks and robots to create a balanced assembly line. Using the standard PSO, two heuristics are solved and the performances are compared with the benchmark results. IBM Cplex Optimization studio Version 12.6.0.0 is also used to solve the problems. Only fourteen problems out of thirty two benchmark datasets could be solved using Cplex. When comparing the results obtained using recursive and consecutive, it is observed that consecutive allocation procedure performs better in terms of quality of solution. Hence, consecutive allocation procedure is used to solve four variants of PSO and hybrid PSO models. PSO variants are developed based on the variation in the

velocity update equation of PSO. Hybrid PSO models are developed by hybridizing PSO with other metaheuristics. It is found that the results of PSO and variants of PSO are very close to Cplex solutions which show the effectiveness of the proposed algorithm. Among the four variants and hybrid PSO algorithms, hybrid CS-PSO performs better in terms of the quality of the solution for most of the problems tested. Computational times for the proposed variants are also presented. The complexity of RALB is also presented with various complexity measures.

The chapter also presented a new model for solving a robotic U-shaped assembly line balancing (RUALB) problem for the objective of minimizing cycle time in a U-shaped robotic assembly line. The literature on this problem is very limited. Particle swarm optimization is proposed to solve the problem. The tasks and robot assignment in the U-shaped configuration are highly complex compared to a straight assembly line. Thirty-two benchmark problems originally proposed by earlier researchers to solve RALB-2 were adopted to test the performance. Parameters are chosen based on pilot studies and from the experimental results it is observed that cycle time for U-shaped robotic assembly line is lower than that of the straight robotic assembly line. Different complexity measures are also presented along with the computational time details. The limitations of this work are that only one robot can be assigned to only one workstation. It cannot handle multiple workstations, the workstations in the assembly line cannot split the tasks and the computational time increases significantly when the problem size increases.

CHAPTER 6

Particle Swarm Optimization to Solve Energy Based RALB Problems

Increasing energy cost and need of creating eco-friendly environment manufacturing industries gives importance for reducing energy consumption. Robotic assembly lines are used extensively and these systems are cost intensive. Hence, there is a requirement of efficiently balancing the assembly line by allocating equal amount of work to workstations and assigning the best fit robot to perform the tasks allocated to the workstations. No research could be found on optimizing cycle time and total energy consumption concurrently in robotic assembly line systems to date. The objective of this section is to propose models with dual focus on time and energy to minimize the cycle time and total energy consumption simultaneously, one model (time based model) with the primary focus to optimize cycle time and the other model (energy based model) with the primary focus to optimize total energy consumption.

The models proposed have a significant managerial implication in real assembly line systems. Suitable models could be selected based on the priorities of the management. The two models proposed in this section are very well applicable to automobile body shop with robot based lines. The main objective is to propose an optimization model to optimize time and energy and to solve the model using a heuristic algorithm. Since the problem falls under the category of NP-hard, Particle Swarm Optimization algorithm is used to optimize the objectives of two models proposed. Time based model optimizes the cycle time of the robotic assembly line as the primary objective and the total energy consumption as the secondary objective. Whereas, energy based model optimizes the total energy consumption of the robotic assembly line as the primary objective and cycle time as the secondary objective. Assignment of tasks to workstation and the robot assignment for the workstation is similar to the consecutive allocation procedure.

6.1 Straight RALB - Cycle time and Energy consumption

This section presents the procedure followed to find out the cycle time and energy consumed in a straight robotic assembly line. Cycle time and energy consumed for a

straight robotic assembly line is calculated using time based model and energy based model. The following section, describes in detail the procedure implemented.

6.1.1 Time based model

Using the time based model, tasks are allocated to the workstations and best fit robots are allotted to perform the allocated tasks in a straight robotic assembly line. The energy consumed for performing the allocated tasks is calculated. As explained in section 5.1.1, the tasks in the sequence will be allocated using the consecutive allocation procedure. In this procedure cycle time is calculated. The maximum of the workstation time is the cycle time for the assignment made. Workstation time is the sum of robot processing times of the tasks by the allotted robots. Total energy consumption for the assignment made is calculated. Energy consumption for a workstation is calculated by multiplying the total robot task time by the power consumed by that robot. Total energy consumption for the assignment is obtained by adding the energy consumed by all workstations. For the illustration purpose refer Figure 5.7, Figure 5.8 and Figure 5.9. In the example shown in Figure 5.8, workstations time are calculated and the cycle time is evaluated to be 143. From Figure 5.9, workstation times of different workstations are shown. These workstation times are used to calculate the energy consumption. Workstation times are 143, 136, 115 and 84. Refer Figure 5.9 for the workstation times and Appendix 2 for the power consumption values of the robots. Energy consumption of each workstation is calculated using the workstation times and the power consumption of the robot which is allotted to perform the tasks at the workstation:

Energy consumption at a workstation = Workstation time × Power Consumption of the robot

Energy consumption at Workstation 1 = 143 × 0.35= 50.05 kJ

Energy consumption at Workstation 2 = 136 × 0.35= 47.6 kJ

Energy consumption at Workstation 3 =115 × 0.3=34.5 kJ

Energy consumption at Workstation 4 = 84 × 0.4= 33.6 kJ

Total energy consumption of the assignment= 50.05+47.6+34.5+33.6=165.7 kJ and the cycle time of the assignment is 143. The results obtained for the 11 tasks problem using time based model is shown in Figure 6.1. The figure shows the tasks and robot allocation along with the workstation times and the energy consumption at each workstation.

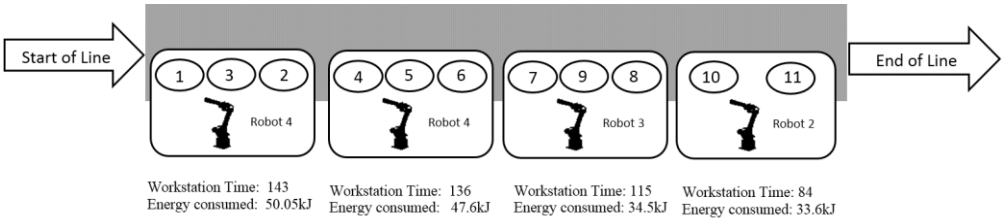

Figure 6.1 Final Solution for time based model in a straight RALB

6.1.2 Energy Based Model for straight robotic assembly line

This procedure is used to calculate the total energy consumption of a straight robotic assembly line. An initial energy consumption of the assembly line, E_0 is used to start the procedure. It is checked if the tasks can be assigned to the workstations until the sum of energy consumption of the tasks is less than or equal to E_0 and that respective robot which consume less energy is assigned to the workstation to perform the tasks. If it is not possible to assign the tasks to the workstation within the given initial E_0 value, E_0 is incremented by *'one'* and the assignment procedure is repeated until all the tasks are assigned to the workstations. Total energy consumption of the assembly line is calculated by adding the energy consumed at each workstation. Stepwise illustration is provided for explaining the task and robot allocation using the energy based model. Precedence graph shown in Figure 5.2 and energy consumption of the robots shown in Table 6.1 are used for the illustration.

Step 1. Feasible sequence of tasks (*1-3-2-4-5-6-7-9-8-10-11*) which meets the precedence constraints is considered for illustration. Initial value of E_0 is the mean of minimum energy consumption of the robots for the tasks. The initial energy consumption (E_0) of the assembly line is calculated using

$$E_0 = [\sum_{j=1}^{N_a} \min_{1 \le i \le N_r} e_{hi} / N_w] \qquad \textbf{(6.1)}$$

Table 6.1 shows the energy consumption of the robots; using this data initial E_0 for the example is found out to be 35.

E_0= [15+15+11+13+9+19+12+10+11+11+15]/4= 35

Step 2. The first station is opened and the procedure tries to allocate the tasks according to the sequence in the order of occurrence, if one or more robot could perform the

allocated tasks within E_0. Each workstation *s* has a set of preferred/ allotted robots *H* which is defined as follows:

$$k \in H, if \;\; m(k) \geq m(h), \text{ where } 1 \leq h \leq N_r \qquad \textbf{(6.2)}$$

where *m (h)* is the maximal number of tasks a robot *h* can perform in the given sequence *sq* within E_0.

$$E_s(h) = \sum_{k=p1_w}^{p1_w+m(h)} e_{h,sq(k)} < E_0 \leq \sum_{k=p1_w}^{p1_w+m(h)+1} e_{h,sq}(k) \qquad \textbf{(6.3)}$$

Next, it defines the robot to be assigned to the workstation *s* as:

$$h(s) = k \;\; \textbf{If} \;\; E_s(k) < E_s(h) \quad \forall h \in H \qquad \textbf{(6.4)}$$

Step 3. The start of position of the next workstation is calculated as follows:

$$p1_{s+1} = pr_s + 1 = p1_s + m(h(s)) + 1 \qquad \textbf{(6.5)}$$

Step 2 is to be repeated until all tasks are assigned to given number of workstations.

Step 4. If the tasks are not possible to be assigned within the given E_0, increment E_0 by *'one'* and repeat the steps 2 and 3 until all the tasks are allotted.

Step 5. The total energy consumption for the assignment is calculated by adding the energy consumption of all workstation.

Step 6. The workstation time of the assignment made is calculated by finding out the time taken by the robot at each workstation. The maximum workstation time is the cycle time of the assignment made.

In the example for energy based model, it is found that tasks 10 and 11 are left unassigned as when E_0=35 as shown in Figure 6.2 a). E_0 is incremented till 43 for accommodating all the tasks in the four workstations and the completed allocation is shown in Figure 6.2 b). The total energy consumption of the assembly line when tasks are allocated based on the energy model is found to be 150 kJ (27+37+43+43). The workstation time of the each workstation is calculated by adding the processing time of the tasks assigned to that workstation for the robot assigned. In this example workstation time of the assignment made is calculated by using the processing time given in Table 5.2

Time at Workstation 1 (*Robot 3*) = 51+38=89

Time at Workstation 2 (*Robot 4*) = 42+40+25=107

Time at Workstation 3 (*Robot 1*) = 77+51+43=171

Time at Workstation 4 (*Robot 1*) = 50+45+76=171

The maximum workstation time is the cycle time of the assignment made and the cycle time is 171. The results obtained for the 11 tasks problem using energy based model is shown in Figure 6.3.

Table 6.1 Energy consumption for 11 tasks by 4 robots

Tasks	Robot 1	Robot 2	Robot 3	Robot 4
1	20	15	15	17
2	27	40	27	15
3	16	32	11	18
4	13	16	27	14
5	23	14	10	9
6	19	26	25	25
7	13	20	12	17
8	13	17	10	15
9	11	30	12	12
10	11	18	12	27
11	19	15	25	30

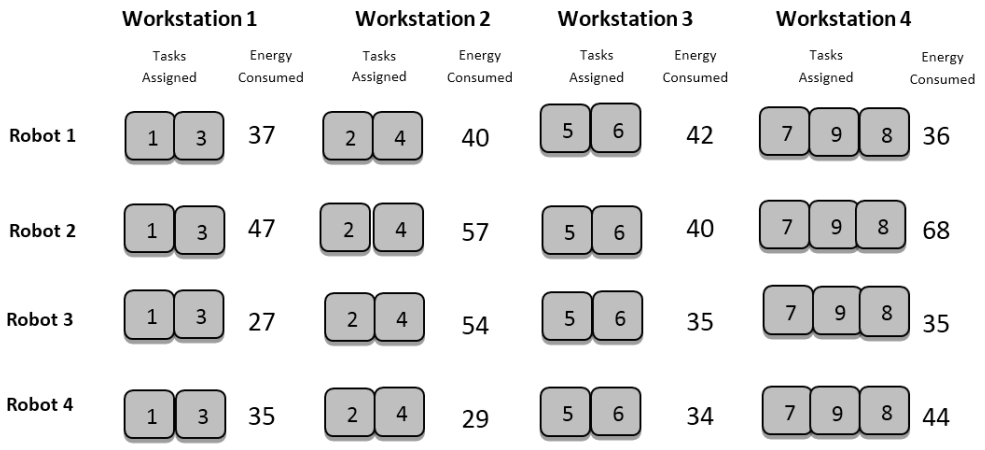

Figure 6.2 a) Task and Robot allocation using energy based model with initial E_0

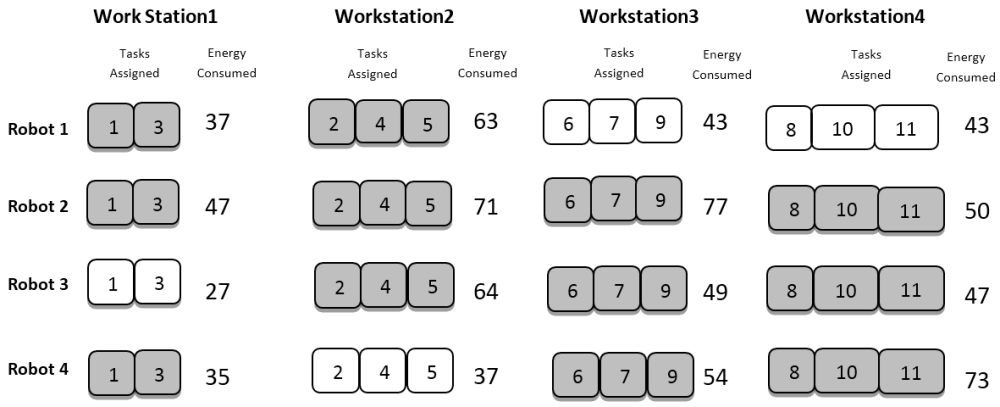

Figure 6.2 b) Completed allocation using energy based model

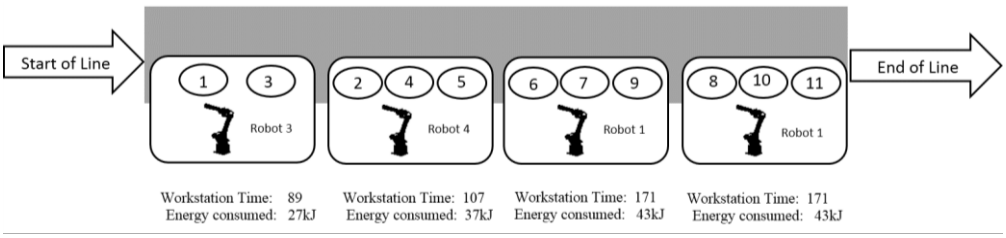

Figure 6.3 Final Solution for the energy based model in a straight RALB

6.1.3 Energy Consumed by Robots during standby mode

Energy consumed by robots or any electrical equipment's when it is switched off or not performing the main function is termed as standby energy. Energy consumption during standby mode is an increasing fraction of energy use in Organization for Economic Cooperation and Development (OECD) countries. Due to the increased usage of new technologies and equipment's there is a growth in standby power usage. It is observed that by reducing the stand by energy consumption worldwide there could be reduction of CO_2 emission by one percent. Energy consumed during standby mode is calculated by assuming that power consumed by the robot during the standby mode is 10% of the original power (Bertoldi et al., 2002). This is an assumption considered in this section to calculate the energy consumed during standby time. An assembly line is considered to be completely balanced if the total slack (i.e., the sum of the stand-by times of all the stations along the line) is as low as possible. However, in practical cases it is difficult to get 100% efficient balanced assembly line. In this section, the energy consumed by the workstations during the (standby mode) is also considered. Standby time is calculated as follows:

Stand by time of workstation= Cycle Time- Workstation time **(6.6)**

The steps involved in calculation of energy consumed in an assembly line are as follows:

1. Calculate Cycle Time of the assembly line.
2. Find the standby time of each workstation.
3. Calculate the standby energy consumed by the robots allotted to the workstation.

Energy is calculated by using $(E=P\times t)$ here P is the power of the robot allotted to the workstation and t corresponds to the standby time of the workstation. Sum up all the standby energy of all the workstations in an assembly line.

Table 6.2 shows an example problem with 9 workstations. Column I show the workstation number, Column II shows the robots assigned after task allocation. Column III and IV shows the workstation time and standby time of the workstation. In this problem cycle time is found to be 110. Using the standby time and power of the robot (Refer Appendix 2), standby time energy is calculated. The standby time energy is evaluated for both the models and it is added to the energy consumption during the production time to get the total energy consumed in an assembly line. Both the models employ the same procedure for the evaluation of the standby energy consumption.

Table 6.2 Standby time Energy Evaluation

Workstation Number	Robot Assigned	Workstation Time	Standby Time	Standby Energy (kJ)
1	4	59	51	2.04
2	7	109	1	0.025
3	4	92	18	0.72
4	9	110*	0	0
5	7	104	6	0.15
6	4	109	1	0.04
7	7	87	23	0.575
8	7	98	12	0.3
9	7	98	12	0.3
*Cycle Time		**Total Standby Energy**		**4.15 kJ**

6.1.4 PSO for solving time and energy based model

PSO algorithm is proposed to solve both the proposed models. PSO algorithm starts with an initial population and initial velocity. Initial population is generated based on the heuristics as explained in Section 5.1.1. Same set of velocity pairs are used for

the velocity update. Pseudo code of PSO as shown in Figure 5.1 is adopted for solving this problem.

Velocity and position of the particles in PSO are updated using Equation 5.1 and 5.2. The equation is reproduced here for the reader's clarity.

Position update is done iteratively using Equation 5.2

$$P_i^{t+1} = P_i^t + v_i^{t+1}$$

The velocity of each particle is updated iteratively Equation 5.1

$$v_i^{t+1} = c_1 v_i^t + c_2 \text{ x } [U_1 \text{ x } ({}^e P_i^t - P_i^t)] + c_3 \text{ x } [U_2 \text{ x } (G - P_i^t)]$$

Where U_1 and U_2 are the velocity coefficients (random numbers between 0 and 1), v_i^t is the initial velocity, ${}^eP^t_i$ is the Local best, G is the Global best and P_i^t is the current particle position, c1, c_2 and c_3 are the acceleration coefficients respectively.

6.1.5 Computational study and Discussions

The computational experiments are conducted in order to test the performance of the two proposed models for straight robotic assembly line (RAL) problem. PSO is proposed to solve the problem. The following section describes the experiments conducted.

6.1.5.1 Datasets for time based model and energy based model

Gao et al. (2009) generated 32 test problems for RALB using 8 precedence graphs available in http://www.assembly-line-balancing.de/ and the time data of 32 test problems are used for evaluation of both the models. Power consumption by robots is randomly generated and it is shown in Appendix 2 for small size datasets and for large size datasets it is shown in Appendix 3. The energy consumption of a task *i* by robot *h* is calculated as follows:

$$E = P_h \times t_{ih} \qquad \textbf{(6.7)}$$

Where, t_{ih} is processing time of the task *i* by robot *h*. P_h is the power consumption of robot *h*. The datasets are divided into two groups: small and large size datasets. Small dataset contains problems with 25 tasks to 70 tasks. Large size datasets consists of problem with tasks 89 tasks to 297 tasks.

6.1.5.2 Parameters of two models proposed

32 test problems are solved for the proposed two models using PSO algorithm. The parameters used in PSO are chosen experimentally in order to get a satisfactory solution quality in an acceptable time span. It is well recognized that the parameter values significantly affect the solution quality. Experiments are performed to find the optimal parameters. Three data sets of different sizes are chosen to find the parameters which yielded best solution. Different combinations of the parameters are tested until the best combination is achieved. Quality of solution is given importance compared to the computational time in selecting the parameters. PSO Parameters chosen to evaluate the two models are shown in Table 6.3

Table 6.3 PSO Parameters selected for evaluating the models

Time Based Model	Energy Based Model
Population size: 25	Population size 25
Number of iterations: 30	Number of iterations:40
Learning coefficients: c_1-1, c_2-1 and c_3-2	Learning coefficients: c_1-1, c_2-2 and c_3-2

6.1.5.3 Performance analysis of two proposed models

The two models are evaluated on the two factors using cycle time and total energy consumption. The total energy consumption and cycle time obtained using the two proposed models are presented in Table 6.4 and Table 6.5. Results obtained from the computational analysis for small size datasets are presented in Table 6.4 and Table 6.5 shows the results for large size datasets. Sum of energy consumption during the production mode and standby mode is the total energy consumption of the assembly line. Graphical comparison of the total energy consumption for both small and large size datasets is presented in Figure 6.4 and Figure 6.5. It is observed from the tables and graphs that the total energy consumption is lower for energy based model and cycle time is lower for the time based model for all the datasets evaluated here.

The average of the difference in the energy consumption between the models is taken to find out the average energy saving. The average energy saving for small size datasets is found to be around 113 kilojoules. The average energy savings for the large size datasets is found to be 1062 kilojoules. Figure 6.6 and Figure 6.7 represents the difference in energy consumption between the time based model and energy based

model and the graph represents the energy saved by using energy based model when compared to the time based model.

Table 6.4 Total energy consumption and cycle time evaluated for small size datasets

Small Size Datasets	Time Based Model		Energy Based Model	
	Total Energy Consumption (kilojoules)	Cycle Time	Total Energy Consumption (kilojoules)	Cycle Time
25-3	514	503	494	641
25-4	347	293	342	314
25-6	420	221	365	235
25-9	265	110	248	142
35-4	1091	341	1072	516
35-5	959	357	929	424
35-7	1180	226	1015	342
35-12	755	105	697	160
53-5	2707	454	2700	587
53-7	2197	293	1989	343
53-10	2513	224	2215	273
53-14	2237	146	2177	200
70-7	4218	446	4146	463
70-10	3228	259	3069	290
70-14	4092	194	3871	290
70-19	3732	139	3323	255

Table 6.5 Total energy consumption and cycle time evaluated for large size datasets

Large Size Datasets	Time Based Model		Energy Based Model	
	Total Energy Consumption (kilojoules)	Cycle Time	Total Energy Consumption (kilojoules)	Cycle Time
89-8	5078	464	5043	562
89-12	6314	317	5683	430
89-16	5191	219	5119	340
89-21	4734	176	4250	206
111-9	4734	526	4250	735
111-13	8207	317	7267	396
111-17	7403	250	6945	280
111-22	7400	185	6909	255
148-10	10166	556	9840	678
148-14	12045	420	10654	461
148-21	11467	272	10131	335
148-29	9290	190	8606	263
297-19	26849	594	25232	809
297-29	26161	428	24970	466
297-38	25450	295	22862	348
297-50	24870	256	22243	348

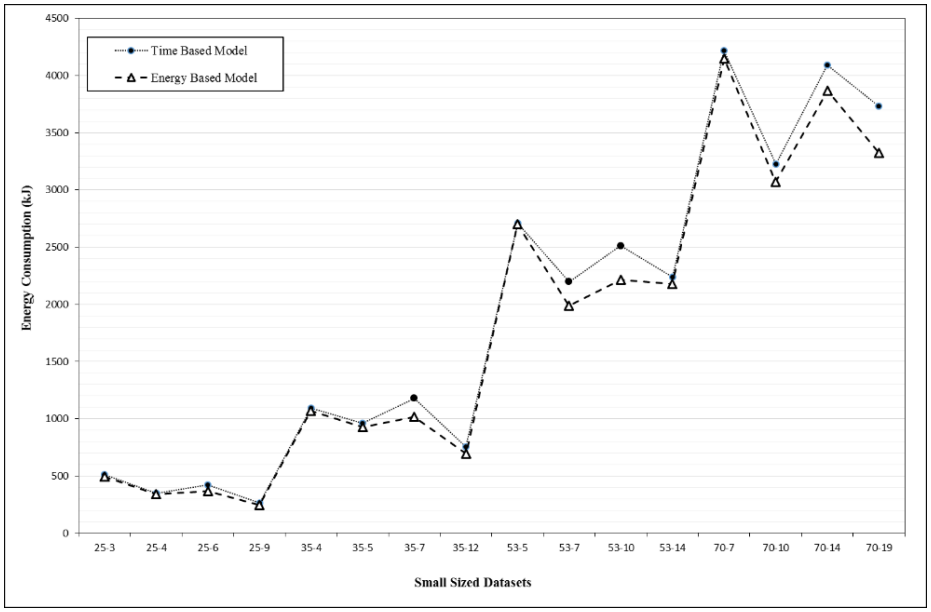

Figure 6.4 Comparison of Energy consumption in small size datasets for two models in straight RALB

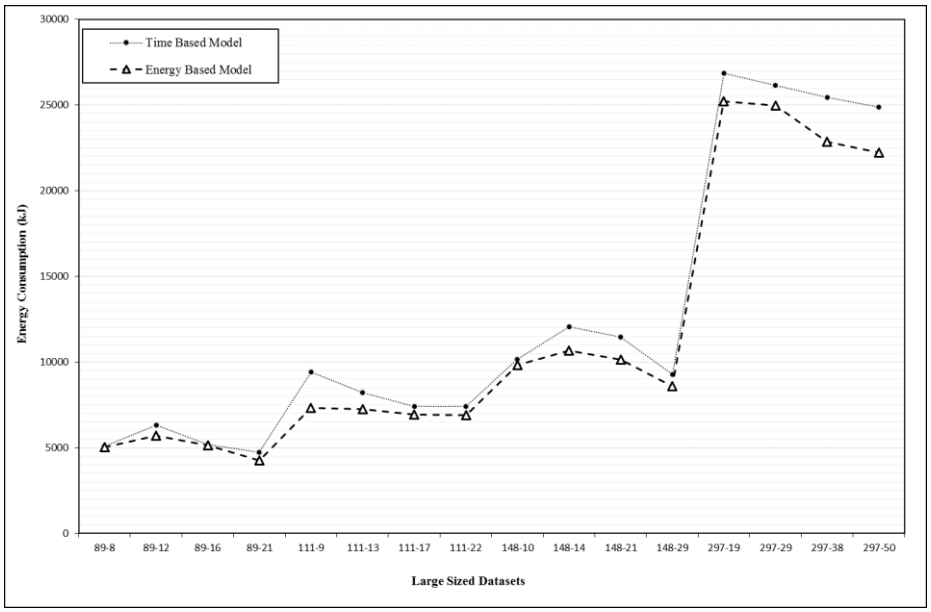

Figure 6.5 Comparison of Energy consumption for large size datasets for two models in straight RALB

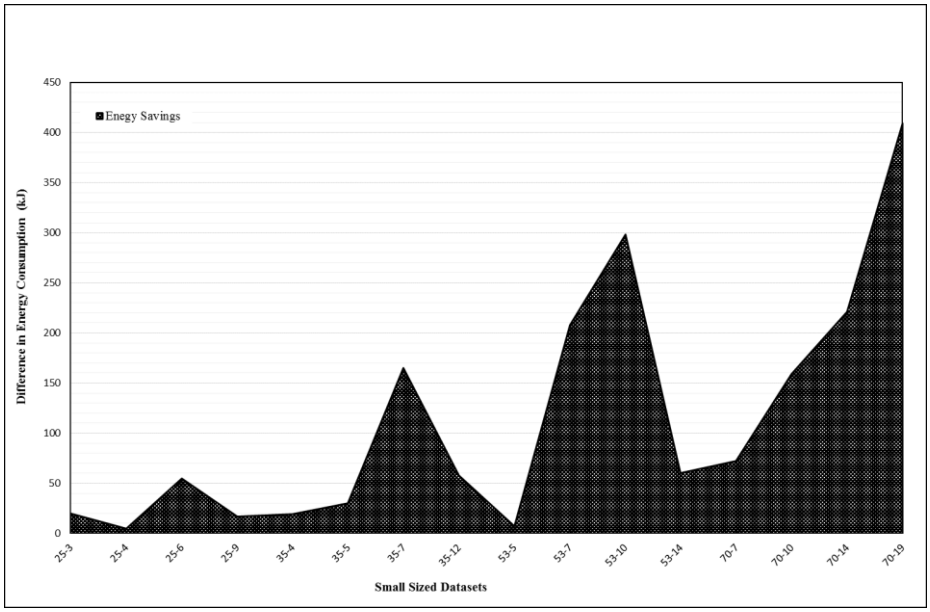

Figure 6.6 Energy saving potential in small size datasets for energy based model in straight RALB

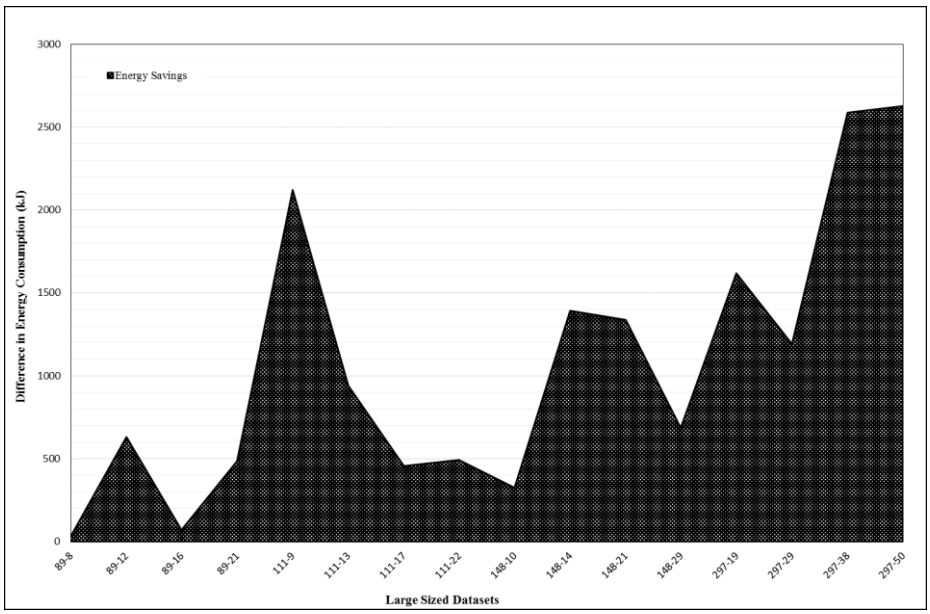

Figure 6.7 Energy saving potential in large size datasets for energy based model in straight RALB

It is observed from the Figure 6.6 that datasets with more amount of tasks, the energy saved is high for most of the problems in the group of large size datasets when energy based model is considered. Figure 6.7 represents the energy saving for large size datasets and it is observed that higher the number of task in the problem evaluated, energy saving is more in case of energy based model. Figure 6.8 and Figure 6.9 shows

the performance of time based model in terms of cycle time. Time based model performs better for both small and large size datasets when compared with the energy based model in case of minimizing the cycle time. Average reduction in cycle time when the time based model is compared with energy based model is found to be 73 units for small size datasets and for large size datasets average cycle time reduction is found to be 109 units.

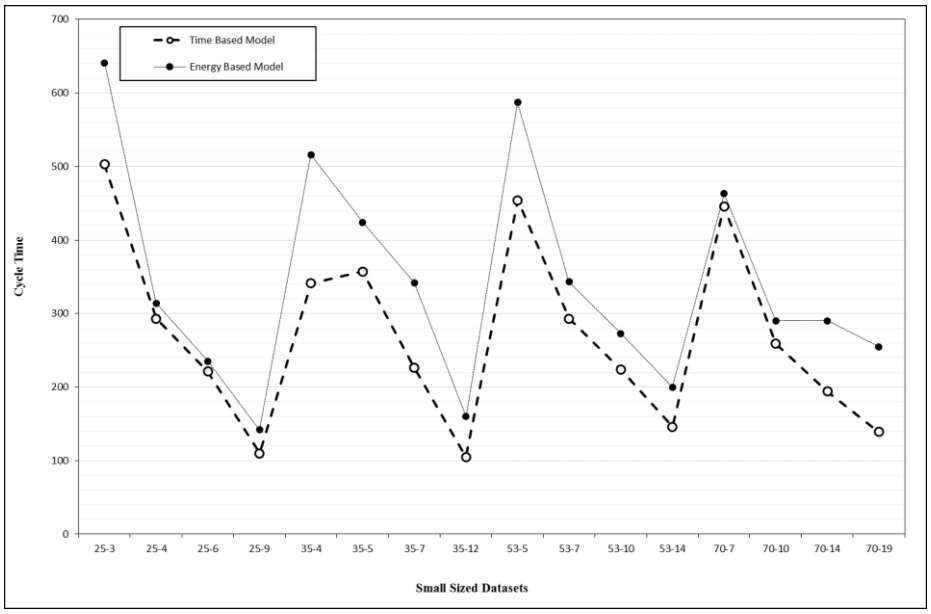

Figure 6.8 Comparison of Cycle Time in small size datasets between two models in straight RALB

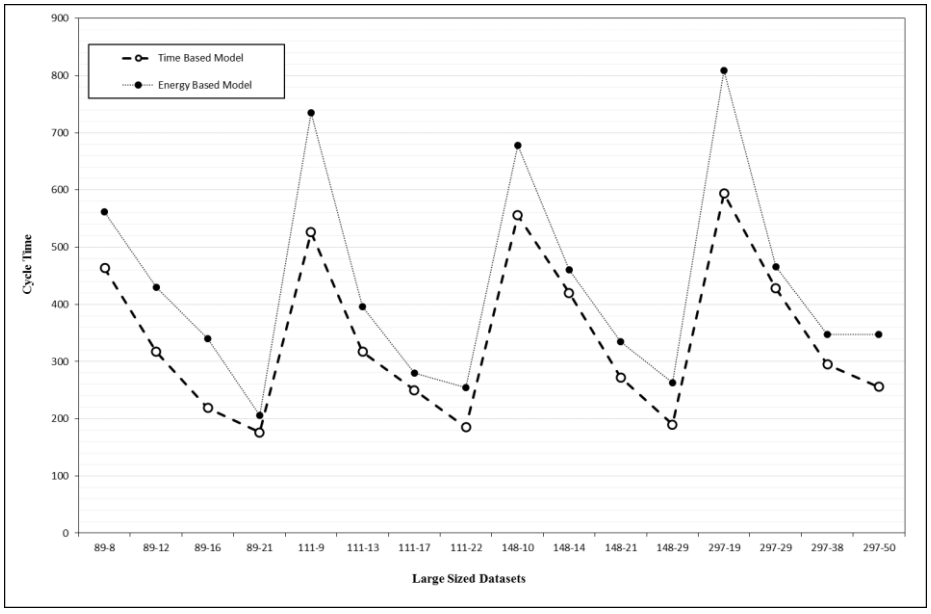

Figure 6.9 Comparison of Cycle Time in large size datasets between two models in straight RALB

Two factors which are very important in a manufacturing setup are: increase the productivity and also to reduce the energy consumption. The objective of the work is to propose two models with dual focus on time and energy consumption in a straight robotic assembly line. Based on the priority of the management, the primary focus between time and energy could vary at different time horizon. The appropriate model could be selected based on the priority. These comparison results could be used for important managerial implications in real life assembly line systems.

6.1.5.4 Computational time

The solution quality is given more importance than the computational time. The average computational time for the time based model and energy based model for the 32 problems evaluated are presented in Table 6.6. Computational results shows that time taken for computing the time based model is very less compared to that of the energy based model for large size datasets.

Table 6.6 Average Computational Time in seconds for the proposed models

Problem Set	Tasks	No. of Problems	CPU Time	
			Time Based Evaluation	Energy Based Evaluation
1	25	4	3	5
2	35	4	8	9
3	53	4	18	21
4	70	4	46	54
5	89	4	62	79
6	111	4	106	130
7	148	4	246	295
8	297	4	1246	1512

6.2 U-shaped RALB - Cycle time and Energy consumption

This section presents the procedure followed to find out the cycle time and energy consumed in a U-shaped robotic assembly line. Cycle time and energy consumed for a U-shaped robotic assembly line is calculated using time based model and energy based model. The following section, describes in detail the procedure implemented for finding the cycle time and energy consumption in a U-shaped robotic assembly line layout by balancing the tasks and robots. Until now, U-shaped robotic assembly line has received very less attention. Besides what is proposed here, it could be seen that no research has been done on optimizing cycle time and energy consumption in a U-shaped robotic assembly line. Two models are proposed to evaluate the cycle time and energy

consumption in an assembly line. These models could be selected based on the demands and priorities of the management. This problem falls under NP-hard; hence PSO is used for solving the two models. Procedures implemented to evaluate the cycle time and energy consumption using the two models are presented here.

6.2.1 Time based model for U-shaped robotic assembly line

Time based model proposed here calculates the cycle time of the U-shaped robotic assembly line by allocating tasks and robots optimally. For the allocation made, the energy consumed by each workstation is calculated. The standby energy consumed is also evaluated. The sum of energy consumed during production mode and standby mode is the total energy consumed by the U-shaped robotic assembly line. As explained in Section 5.5.3, the tasks are allocated to the workstations and best fit robots are allotted to the workstations to perform the allocated tasks in a U-shaped assembly lime. The stepwise procedure involved in calculating the cycle time and energy consumed in U-shaped robotic assembly line is presented here. In this procedure cycle time is calculated. The maximum of the workstation time is the cycle time for the assignment made. Workstation time is the sum of robot processing times of the tasks by the allotted robots. Total energy consumption for the assignment made is calculated. Energy consumption for a workstation is calculated by multiplying the total robot task time by the power consumed by that robot. Total energy consumption for the assignment is obtained by adding the energy consumed by all workstations. In the example shown in Figure 5.29, workstations time are calculated and the cycle time is evaluated to be 121. The workstation times are used to calculate the energy consumption. Workstation times are 121, 115, 107 and 116. Using the power consumption for 11 task problem presented in Appendix 2, the energy consumption of each workstation is calculated using the workstation times and the power consumption of the robot which is allotted to perform the tasks at the workstation:

Energy consumption at a workstation= Workstation time× Power Consumption of the robot

Energy consumption at Workstation *1*=121× 0.4= 48.4 kJ

Energy consumption at Workstation *2*= 115× 0.3= 34.5 kJ

Energy consumption at Workstation 3=107× 0.35=37.45 kJ

Energy consumption at Workstation 4= 116× 0.4= 46.4 kJ

Total energy consumption of the assignment= 48.4+34.5+37.45+46.4= 166.75 kJ and the cycle time of the assignment is 121. The results obtained for the 11 tasks problem using time based model is shown in Figure 6.10. The figure shows the tasks and robot allocation along with the workstation times and the energy consumption at each workstation. The energy consumed during the standby mode is calculated using the workstation times and it is found to be 0.95 kJ. The total energy consumption for the U-shaped robotic assembly line using time based model is the sum of energy consumption during the production mode and energy consumption during the standby mode and the total energy consumption is 167.7 kJ.

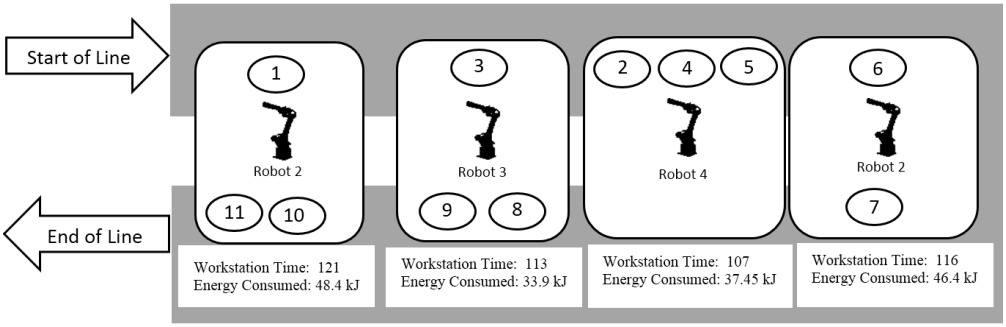

Figure 6.10 Solution for the time based model in a U-shaped RALB

6.2.2 Energy Based Model for U-shaped robotic assembly line

This procedure is used to calculate the total energy consumption of a U-shaped robotic assembly line. Task allocation procedure is different from the straight robotic assembly line allocation. This procedure also starts with an initial energy consumption of the assembly line. In U-shaped robotic assembly line, tasks are allocated to the workstation by moving forward and backward through the precedence diagram in contrast to a typical forward move in the traditional assembly systems. Best robot which can perform the tasks allotted to the workstations with minimum energy consumption are checked for allotment. The procedure tries to allocate the maximum number of tasks to the workstations without violating the precedence constraints. If the initial E_0 cannot accommodate all the tasks, E_0 is incremented by one and the procedure is repeated to accommodate all the tasks. The allocation done gives the energy consumption at each workstation. The time taken by each workstation is also calculated which is used to find the cycle time and standby time for each workstation. Using the standby time for each

workstation, the energy consumed during the standby mode is calculated. The total energy consumed by the U-shaped assembly line is the sum of energy consumption during production mode and standby mode. An illustration is provided in this section which explains the task and robot allocation and calculation of energy consumption in a U-shaped robotic assembly line. Sequence of tasks which meets the precedence constraints is considered for illustration. Let, the sequence be, ***1*-3-2-4-5-6-7-9-8-10-11.** 11 task and 4 workstation problem is considered for the illustration. Energy consumption details of each tasks and robots are presented in Table 6.1.

Step 1. E_0 is calculated and it is found to be 35 using Equation 6.1.

Step 2. For the initial E_0, the procedure tries to allocate the tasks to the workstations starting from the first workstation. Procedure allocates the maximum tasks by checking either side of the sequence if any of the robots could perform the tasks within E_0. Since it is U-shaped, search space is more due to different possible combinations

Step 3. If the previous workstation cannot accommodate more tasks, the next workstation is allocated with the remaining tasks.

Step 4. After trying with initial E_0, if the procedure still was not able to assign all the tasks within the E_0, E_0 is increment by 'one' and repeat the Step 2 and 3 until all tasks get assigned to the workstation.

Step 5. Robots are allotted to each workstation with certain set of tasks. Robot which performs the allotted tasks with minimum energy consumption is selected and allocated.

Step 6. Using the robot allocated and tasks in the workstation, the workstation time is calculated for the tasks allocated.

Step 7. The workstation with the maximum workstation time is the cycle time of the allocation.

Step 8. Using the workstation times, the standby time of each workstation is calculated and the energy consumed during the standby mode is calculated using the power consumption of the robots allotted to each workstation.

Step 9. Sum of the energy consumption during production mode and energy consumption during the standby mode gives the total energy consumption of the assembly line.

For energy based model in U-shaped robotic assembly, when the allocation was attempted with initial E_0 it was found that tasks 6 and 7 are left unassigned. E_0 is incremented till 47 for accommodating all the tasks in the four workstations and the complete the allocation. The energy consumption of the assembly line during the production mode when tasks are allocated based on the energy model is found to be 151 kJ (30+33+41+47). The workstation time of the each workstation is calculated by adding the processing time of the tasks assigned to that workstation for the robot assigned. In this example workstation time of the assignment made is calculated by using the processing time given in Table 5.2

Time at Workstation *1(Robot 2)* = 37+38=75, Time at Workstation *2(Robot 3)* = 38+34+41=113

Time at Workstation 3*(Robot 4)* = 42+40+33=115 and Time at Workstation 4*(Robot 3)* = 33+83+40=156. The maximum workstation time is the cycle time of the assignment made and the cycle time is 156. The results obtained for the 11 tasks problem using energy based model is shown in Figure 6.11.

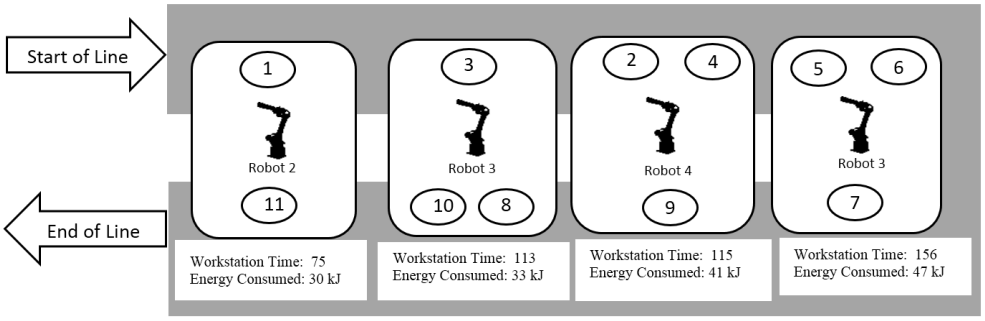

Figure 6.11 Solution for the energy based model in a U-shaped RALB

6.2.3 PSO to solve proposed models in U-shaped RALB

PSO is used for solving the proposed model with an objective of minimizing the cycle time (time based model) and total energy consumption (energy based model) for U-shaped robotic assembly line. The computational experiments are conducted in order

to test the performance of the two proposed models. The following section describes the experiments conducted.

6.2.3.1 Datasets for time based model and energy based model

Thirty two test problems for RALB proposed by Gao et al. (2009) are used for evaluating the two models to solve energy consumption and cycle time in a U-shaped robotic assembly line. Power consumption of robots which are used for the dataset generation is presented in Appendix 2 for small size datasets (up to 70 task problems) and for large size datasets (89 tasks to 297 tasks problem) power consumption details are presented in Appendix 3.

6.2.3.2 Parameter settings for PSO

Pilot studies have been conducted on three different datasets of different sizes to arrive at the best value of population size in the proposed PSO. From the studies it is found that both the proposed models works well with a swarm size of 25 for minimizing energy consumption for the energy based model and minimizing cycle time for the time based model. The acceleration coefficients are also selected by trying different combinations for c_1, c_2 and c_3. Similarly, studies have been carried out to find the suitable stopping condition and the best values are 25 iterations in time based model and for energy based model the total number of iterations is 30. Table 6.7 summarizes the parameters of PSO used to solve the proposed models. During the experimental study, quality of solution is given importance compared to the computational time.

Table 6.7 Parameters of PSO selected for evaluating the two models

Time Based Model	Energy Based Model
Population size: 25	Population size 25
Number of iterations: 30	Number of iterations:40
Learning coefficients: c_1-1, c_2-2 and c_3-2	Learning coefficients: c_1-1, c_2-2 and c_3-2

6.2.3.3 Performance analysis of two proposed models

The performances of two models are evaluated to find out the cycle time and energy consumption. The proposed models are coded in C++ and the performances are tested on Intel core i5 processor (2.3 GHz). The datasets are divided into two groups: small size datasets and large size datasets. The complete details of the results obtained by using the time based and energy based model for small size datasets are presented in Table 6.8 and Table 6.9 shows the results obtained for the two models for the large

size datasets. The energy consumption reported here is the sum of energy consumption during the production mode and energy consumption during the standby mode.

Table 6.8 Results of performance evaluation of two models for small size datasets in U-shaped RALB

Small size Datasets	Time Based Model		Energy Based Model	
	Total Energy Consumption (kJ)	Cycle Time	Total Energy Consumption (kJ)	Cycle Time
25-3	546	500	496	546
25-4	333	293	345	320
25-6	361	188	359	213
25-9	281	109	246	157
35-4	1094	355	1042	509
35-5	1003	333	861	360
35-7	1157	221	1005	307
35-12	743	103	651	126
53-5	2713	443	2665	452
53-7	2111	286	1982	330
53-10	2640	220	2172	252
53-14	2139	144	2039	183
70-7	4410	442	4257	557
70-10	2715	264	3050	294
70-14	4149	194	3845	275
70-19	3754	139	3229	189

Figure 6.12 and Figure 6.13 presents the energy consumption obtained using the time based model and energy based and it is evident from the figure that energy based model is better in terms of minimizing energy consumption when compared with time based model for both the groups of datasets. However, two datasets in the small size datasets (25-4 and 70-10) did not get lower energy consumption using energy based model.

Table 6.9 Results of performance evaluation of two models for large size datasets in U-shaped RALB

Large size Datasets	Time Based Model		Energy Based Model	
	Total Energy Consumption (kJ)	Cycle Time	Total Energy Consumption (kJ)	Cycle Time
89-8	5472	481	4826	532
89-12	5978	315	5665	420
89-16	5221	218	4969	322
89-21	4668	169	4218	201
111-9	9440	522	7230	688
111-13	7936	316	7167	377
111-17	7309	256	6861	265
111-22	6974	181	6800	228
148-10	11127	619	9828	671
148-14	11552	417	10506	458
148-21	11103	270	10079	318
148-29	8576	187	8415	198
297-19	25821	591	24658	697
297-29	25787	390	24666	448
297-38	25061	293	22446	345
297-50	24116	222	22022	341

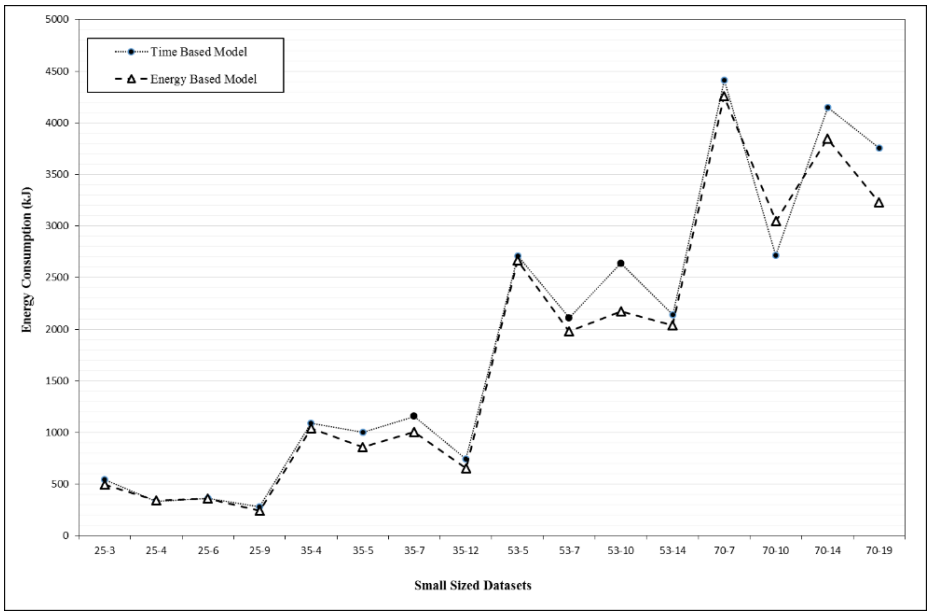

Figure 6.12 Comparison of energy consumption for small size datasets in U-shaped RALB

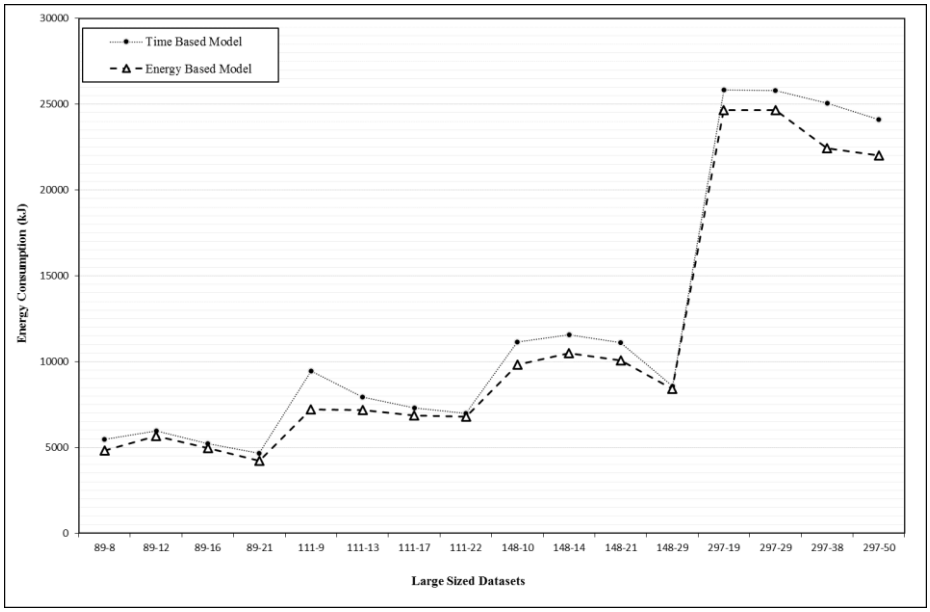

Figure 6.13 Comparison of energy consumption for large size datasets in U-shaped RALB

Energy consumption for energy based model is lower when compared to energy consumption obtained for time based model. Differences in energy consumption between two models are taken and the average of the difference is taken to calculate the average savings. It is observed the average energy saving by using energy based model for small size datasets is around 119 kilojoules. The average energy which can be saved by using energy based model over time based model for large size datasets is found to be 987 kilojoules. Figure 6.14 and Figure 6.15 represents the difference in energy consumption (energy saving potential) between the time based model and energy based model and the graph represents the energy saved by using energy based model when compared to the time based model. From Figure 6.14 it is observed that two problems from the datasets are not able to produce better results for energy based model and they are represented as negative peaks in the graph.

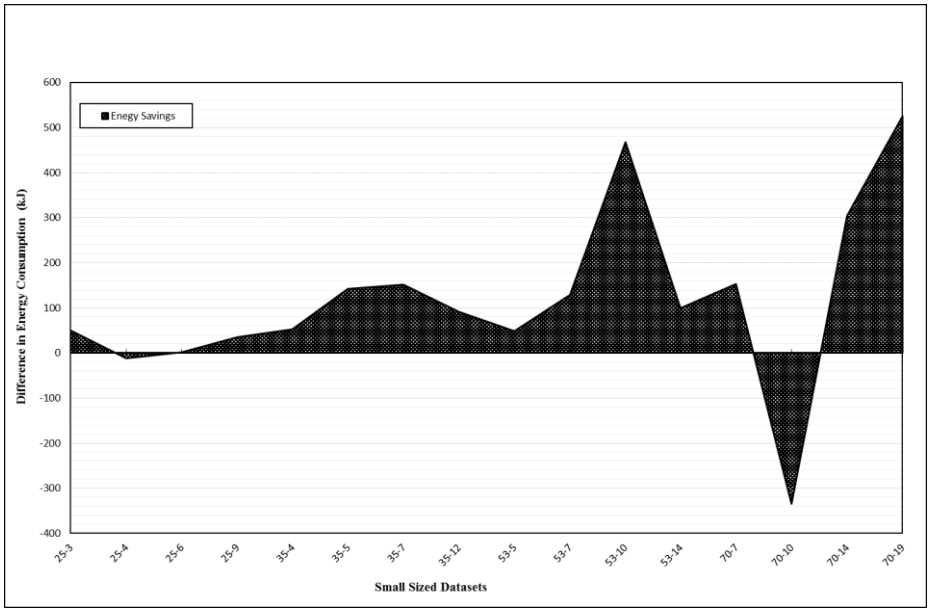

Figure 6.14 Energy saving potential in small size datasets for energy based model in U-shaped RALB

Figure 6.16 and Figure 6.17 represent the comparison for the cycle time evaluated by using the two models. And it is clearly evident that cycle time of time based model is lower when compared to the cycle time obtained using energy based model for all the thirty problems evaluated. The average reduction in cycle time by using time based model for small size datasets is 52 cycle time units and for large size datasets average reduction in cycle time is 66 units.

The manufacturing companies give utmost importance to increase the productivity and reduce the energy consumption. The objective of the work presented here is to propose two models which focus on time and energy consumption concurrently in a U-shaped robotic assembly line. Based on the demands and requirements of the managements, industrial managers can chose any of the two proposed models for applying it in real life assembly line systems.

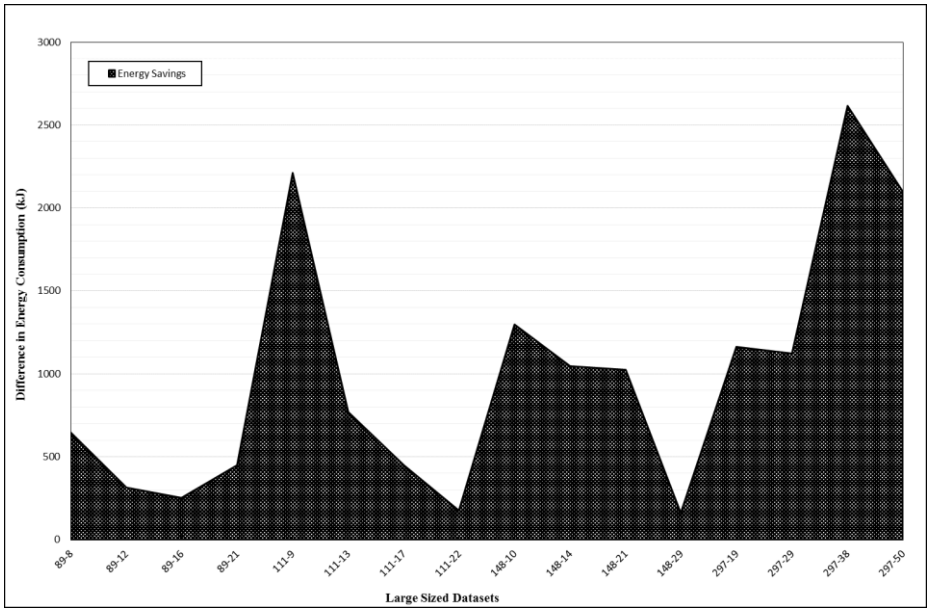

Figure 6.15 Energy saving potential in large size datasets for energy based model in U-shaped RALB

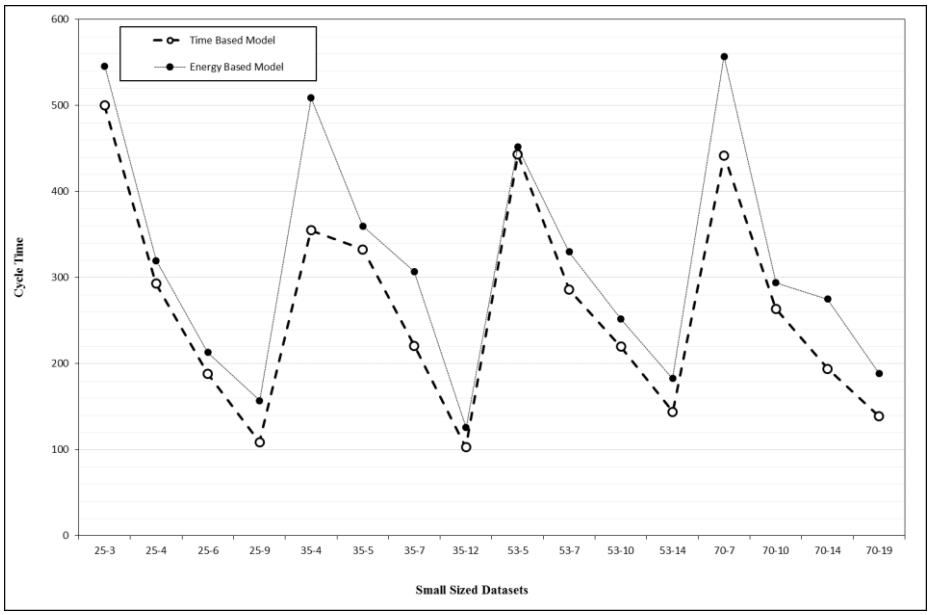

Figure 6.16 Comparison of cycle time obtained for small size datasets in U-shaped RALB

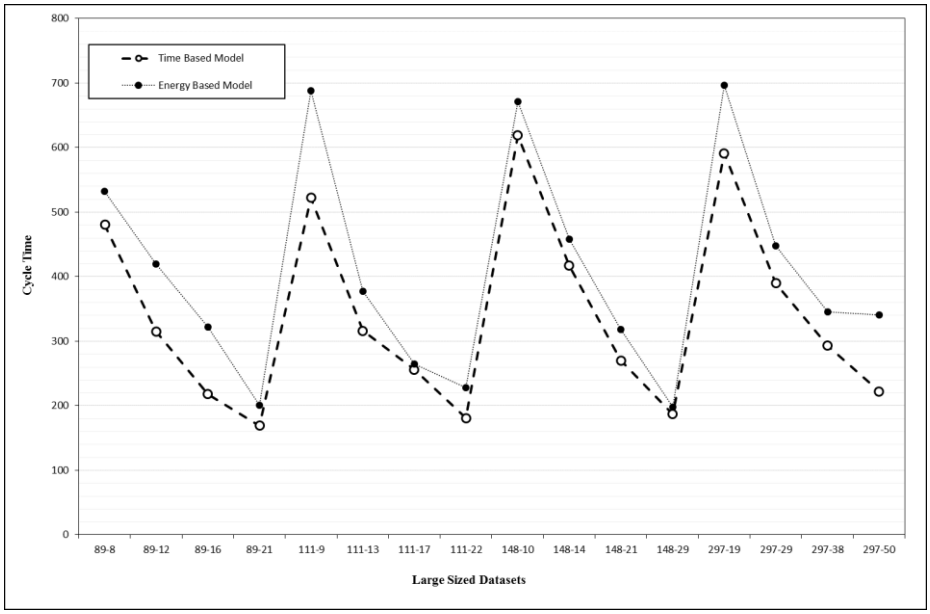

Figure 6.17 Comparison of cycle time obtained for large size datasets in U-shaped RALB

6.2.3.4 Computational time

The average computational time for the time based model and energy based model for the 32 problems of U-shaped robotic assembly line evaluated are presented in Table 6.10. Computational results shows that time taken for computing the time based model is very less compared to that of the energy based model for large size datasets.

Table 6.10 Average Computational Time for the proposed two models

Problem Set	Tasks	No. of Problems	CPU Time	
			Time Based Evaluation	Energy Based Evaluation
1	25	4	10	13
2	35	4	24	25
3	53	4	38	39
4	70	4	72	75
5	89	4	93	94
6	111	4	287	354
7	148	4	559	676
8	297	4	1726	1854

The time consumed for large size datasets from 111 tasks is very high for energy based model when compared to time based model. However, when comparing the time consumed to solve U-shaped robotic assembly line is higher when compared with the computational time for evaluating straight robotic assembly line problems. This is due to large search space and different possible combinations in U-shaped layout.

6.2.4 Comparison of straight and U-shaped RALB

The total energy consumption and cycle obtained using energy based model and time based model for straight and U-shaped robotic assembly line are compared. Table 6.11 is formed by extracting the results from Table 6.4, Table 6.5, Table 6.8 and Table 6.9. The results indicate that energy consumption is very low for U-shaped robotic assembly line when compared to the energy consumption in straight robotic assembly line. Twenty nine out of thirty two datasets yielded lower energy consumption for U-shaped robotic assembly line. Average energy savings which is achieved by using U-shaped robotic assembly line over straight robotic assembly line is calculated as 26 kilojoules for small size datasets (up to problem number 16) and 169 kilojoules for large size datasets.

Table 6.11 Comparison: energy consumption between straight & U-shaped RALB

Problem No:	Problem Dataset	Energy Consumption (kJ)		Problem No:	Problem Dataset	Energy Consumption (kJ)	
		Straight RALB	U-shaped RALB			Straight RALB	U-shaped RALB
1	25-3	**494**	496	17	89-8	5043	**4826**
2	25-4	**342**	345	18	89-12	5683	**5665**
3	25-6	365	**359**	19	89-16	5119	**4969**
4	25-9	248	**246**	20	89-21	4250	**4218**
5	35-4	1072	**1042**	21	111-9	7307	**7230**
6	35-5	929	**861**	22	111-13	7267	**7167**
7	35-7	1015	**1005**	23	111-17	6945	**6861**
8	35-12	697	**651**	24	111-22	6909	**6800**
9	53-5	2700	**2665**	25	148-10	9840	**9828**
10	53-7	1989	**1982**	26	148-14	10654	**10506**
11	53-10	2215	**2172**	27	148-21	10131	**10079**
12	53-14	2177	**2039**	28	148-29	8606	**8415**
13	70-7	**4146**	4257	29	297-19	25232	**24658**
14	70-10	3069	**3050**	30	297-29	24970	**24666**
15	70-14	3871	**3845**	31	297-38	22862	**22446**
16	70-19	3323	**3229**	32	297-50	22243	**22022**
Average Energy Savings for U-shaped			**26 kJ**	**Average Energy Savings for U-shaped**			**169 kJ**

Cycle time of both straight and U-shaped robotic assembly line obtained using time based model are extracted and the results are presented in Table 6.12.

Table 6.12 Comparison: cycle time between straight & U-shaped RALB

Problem No:	Problem Dataset	Cycle Time		Problem No:	Problem Dataset	Cycle Time	
		Straight RALB	U-shaped RALB			Straight RALB	U-shaped RALB
1	25-3	503	**500**	17	89-8	**464**	481
2	25-4	**293**	**293**	18	89-12	317	**315**
3	25-6	221	**188**	19	89-16	219	**218**
4	25-9	110	**109**	20	89-21	176	**169**
5	35-4	341	355	21	111-9	526	**522**
6	35-5	357	**333**	22	111-13	317	**316**
7	35-7	226	**221**	23	111-17	**250**	256
8	35-12	105	**103**	24	111-22	185	**181**
9	53-5	454	**443**	25	148-10	**556**	619
10	53-7	293	**286**	26	148-14	420	**417**
11	53-10	224	**220**	27	148-21	272	**270**
12	53-14	146	**144**	28	148-29	190	**187**
13	70-7	446	**442**	29	297-19	594	**591**
14	70-10	259	**264**	30	297-29	428	**390**
15	70-14	**194**	**194**	31	297-38	295	**293**
16	70-19	139	**139**	32	297-50	256	**222**
Average Percentage Reduction in cycle time using U-shaped			**1.80%**	**Average Percentage Reduction in cycle time using U-shaped**			**1.07%**

From Table 6.12, it is observed that cycle time of U-shaped robotic assembly line obtained using the time based model is lower than the cycle time for straight robotic assembly line problems for 28 out of 32 problems evaluated. The average percentage reduction in cycle time by using U-shaped layout for the small size datasets is computed as 1.8% for small size datasets and the average percentage reduction in cycle time for large size datasets is computed as 1.07%. Hence it is concluded from this study that U-shaped robotic assembly line performs better than straight robotic assembly line for the objective of minimizing cycle time as well as minimizing energy consumption.

6.3 Summary

In manufacturing systems, optimizing cycle time and energy consumption is a very important problem. Reducing energy consumptions helps to improve the productivity and manufacturing companies give importance due to the serious environmental impacts and rising energy cost. Creating an eco-friendly manufacturing system by

minimizing energy consumption is very important in present day context. In this chapter, a study on robotic assembly line balancing problem with an objective of minimizing cycle time and energy consumption simultaneously is considered for two layouts of robotic assembly line (straight and U-shaped).

The work presented in this chapter is an important addition to the literature where majority of the work on robotic assembly line dealt with the objective of minimizing cycle time. A heuristic is developed for minimizing cycle time and total energy consumption in both the layouts of robotic assembly lines. A particle swarm optimization algorithm is proposed to solve the proposed models. Thirty two datasets available in the literature has only time and precedence information. The energy data is embedded into the existing datasets for developing datasets to solve the proposed model. The objective of this work is to propose models with the dual focus on time and energy. The computational experiments are conducted on the two models proposed in this chapter for both the layouts. Energy saving potential for the proposed models is also studied. Depending upon the priority of the management, the primary focus between time and energy could vary at different time horizon. The appropriate model could be selected based on the priority of the management. A comparative study is conducted for the results obtained for two layouts (straight and U-shaped) and it is concluded from the experimental results that U-shaped robotic assembly line performs better in terms of minimizing energy consumption and cycle time when compared with straight robotic assembly line for most of the datasets.

CHAPTER 7

Particle Swarm Optimization & Differential Evolution to Solve Cost Based RALB Problems

In today's competitive world, reducing the cost of the manufacturing component of production is on the mind of manufacturers all over the globe. In a manufacturing scenario, assembly is one of the most important processes. In an assembly line robots are widely used instead of manual labor. By using robots, cost incurred due to manual labor like salary, employee management and safety are eliminated. By employing robots the companies can reduce the direct and overhead costs. In this work, a new robotic assembly line balancing (RALB) problem is developed with an objective of minimizing the total production cost of an assembly line by allocating tasks to the workstations and assigning the cost efficient robot available. PSO and DE are used to solve the problem.

7.1 Straight RALB - Minimizing Assembly Line Cost

This section presents the procedure followed to find out the total assembly line cost in a straight robotic assembly line (RAL). Most of the researchers considered only the objective of minimizing the cycle time in a robotic assembly line. Consecutive allocation procedure is adopted for task and robot allocation with an objective of minimizing the total assembly line cost.

7.1.1 Consecutive Allocation procedure- Straight line

Aim of this allocation procedure is to assign tasks to the workstations and allocate the best fit robot which performs the task with minimum performance cost. The procedure starts with an initial assembly line cost. The tasks are allocated to the workstation within the initial assembly line cost. The initial assembly line cost is determined using Equation 7.1. The procedure tries to allocate the maximum tasks to each workstation for the initial assembly line cost. If the procedure cannot find the optimal allocation within the initial value, the initial value is incremented and the procedure is repeated until all the tasks get assigned.

$$\text{Initial assembly line cost } P_0 = \left\lceil \sum_{j=1}^{N_a} \min_{1 \le i \le N_r} c_{i,j} / N_w \right\rceil \quad \textbf{(7.1)}$$

Following steps are involved in consecutive assignment procedure with an example task sequence ***(1-4-5-3-7-9-2-6-8-10-11)***:

Step 1. Minimum cost to perform each task by any robot among the given set of robot is used to calculate the initial value of P_0. In the given example below initial P_0 is found out to be 98 (refer Table 7.1).

P_0= [33+40+35+36+24+57+37+31+31+36+33]/4=98.

Step 2. For the calculated P_0, the procedure tries to allocate the first task to the first workstation and checks if any of the robot can perform the task within the initial assembly line cost.

Step 3. If yes, the next immediate task in the sequence is allotted to the same work station and checks for the robot to perform those tasks with in the P_0.

Step 4. Tasks are further added to the same station until the cost value exceeds the initial P_0 value.

Step 5. If further tasks cannot be assigned to the workstation next workstation is opened and tasks are allotted.

Step 6. Repeat this procedure until all the tasks are allotted and robots are assigned.

Step 7. If tasks are left unassigned within the initial P_0, P_0 is incremented by '1' and procedure is repeated until all tasks get allotted.

Step 8. The workstation with the tasks allotted is allotted with the best robot which performs the allotted tasks with minimum performance cost.

Step 9. The overall assembly line cost is calculated by summing up the cost of performing the allotted task in each workstation by the allocated robots.

Using the performance cost and precedence relations data presented in Table 7.1, the given sample sequence is evaluated. Figure 7.1 shows the allocation of tasks when P_0 is 98 and it is observed that tasks 6,8,10 and 11 are left unassigned. P_0 is incremented till 110 for the complete allocation as shown in Figure 7.2. The total assembly line cost

is calculated by summing the cost to perform the allotted tasks at each workstation and for the sample problem the total assembly line cost is 429.

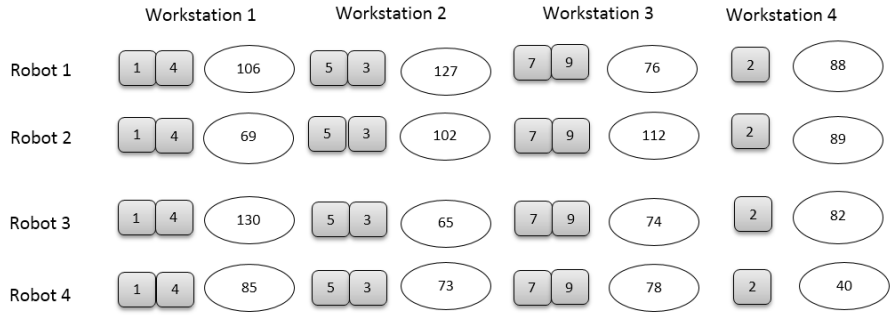

Figure 7.1 Allocation done for initial assembly line cost

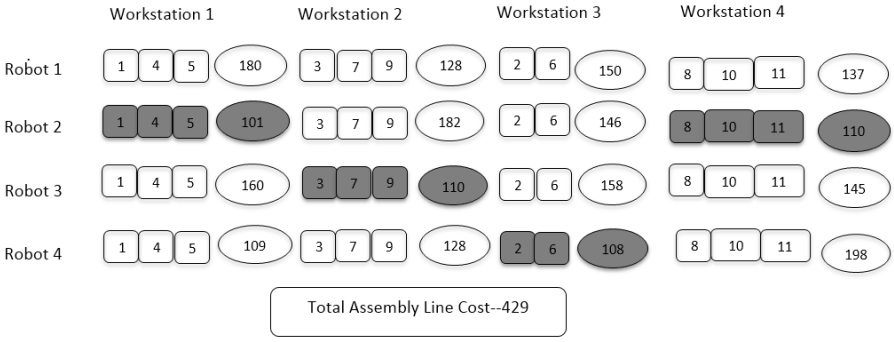

Figure 7.2 Final allocation of tasks and robots using consecutive allocation procedure in straight RALB

Table 7.1 Performance cost data and precedence relations for 11 task problem

Task	Precedence Relations	Cost for performing the tasks				Average Cost
		R1	R2	R3	R4	
1	-	65	33	47	47	48.0
2	1	88	89	82	40	75.0
3	1	52	70	35	50	52.0
4	1	41	36	83	38	50.0
5	1	74	32	30	24	40.0
6	2	62	57	76	68	66.0
7	3,4,5	41	45	37	47	42.0
8	6	40	37	31	42	38.0
9	7	35	67	38	31	43.0
10	8	36	40	38	73	47.0
11	9,10	61	33	76	83	63.0

7.1.2 PSO variants and DE to solve cost based model in straight RALB

Since this problem is a well-known NP-hard, four variants of particle swarm optimization algorithm and Differential Evolution (DE) algorithm are proposed to solve the consecutive allocation procedure. Four variants as explained in Section 5.2 are implemented to solve the problem. Differential Evolution algorithm is also developed to solve the problem. This section gives the details on how DE is implemented.

7.1.2.1 Differential Evolution

Storn and Price (1997) proposed a simple algorithm for optimization and engineering problems called Differential Evolution (DE). DE is used to solve different optimization problems and it is reported that DE outperforms other popular evolutionary algorithms (Ali and Törn, 2004, Kaelo and Ali, 2006). Problems with discrete decision variables such as machine layout problem (Nearchou, 2006), and flow-shop scheduling problems (Nearchou and Omirou, 2006) have been tested with DE and better results are reported. Implementation of DE on solving the cost based robotic assembly line problem is explained in the following sections. The algorithm starts with an initial set of random population (target vectors). In DE mutation and crossover are done first before selection process. While in GA, selection process come first and follows by crossover and mutation. A new set of vectors called donor vectors are created using target vectors. A crossover operation is carried out between target vectors and the corresponding donor vectors to generate trial vectors. The selection operation is done by comparing the fitness values of each target vectors and trial vectors. If the trial vector has a better fitness then target vector then it will be selected into the population otherwise target vector will be selected. The above mentioned three processes are repeated until the termination condition is satisfied. Figure 7.3 shows the flowchart of DE.

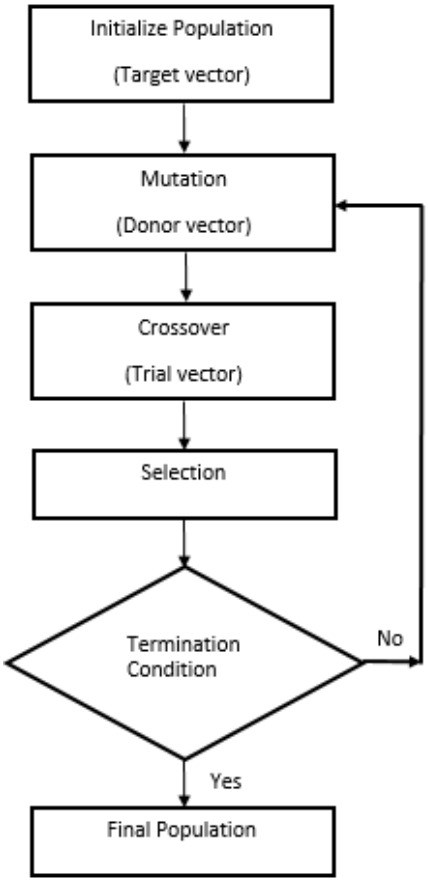

Figure 7.3 Flowchart for differential evolution

The details of how each of the steps is implemented to solve the proposed problem are presented here.

a) Initial Population

DE procedure starts with the initial set of population called as 'target vectors'. Each member (vector) of this population encodes a potential solution for the problem. The vector represents a sequence of numbers (tasks) arranged such a way that it meets the precedence relationship. Based on the heuristics used for generating the initial population for PSO in section 5.1.1, the same heuristics are used for generating initial population in DE for solving the cost based model RALB problems

b) Mutation

In DE there are different variants for mutation operator (Qin and Suganthan, 2005). Only one variant is selected and implemented in this thesis. A population of donor vectors is created by perturbing the population of target vectors. Perturbation is

performed by adding the difference between two randomly selected target vectors to a third target vector which is given in the

$$y_{ig} = x_{r1,g} + F(x_{r2,G} - x_{r3,G}), where\ i\ = 1,...5 \quad \textbf{(7.2)}$$

F is known as the mutation scale factor. For an example let us consider three vectors:

$x_{r1,G}$={1,2,6,3,4,5,7,8,10,9,11}, $x_{r2,G}$={1,2,3,4,5,6,7,8,9,10,11} and
$x_{r3,G}$={1,2,3,6,5,4,7,8,10,9,11}

y_{ig} = {1,2,6,3,4,5,7,8,10,9,11}+F*{1,2,3,4,5,6,7,8,9,10,11}-{1,2,3,6,5,4,7,8,10,9,11}

The pairs of transpositions to get $x_{r3,G}$ from $x_{r2,G}$ are identified. Then apply the mutation factor, the number of pairs are selected and these pairs are used to transposition the values in $x_{r1,g}$ with F=0.5, y_{ig} is generated as explained below.

y_{ig}={1,2,6,3,4,5,7,8,10,9,11}+0.5*(3,5)(8,9)
={1,2,6,3,4,5,7,8,10,9,11}+(8,9)={1,2,6,3,4,5,7,8,9,10,11}

c) Crossover

Once the mutation phase is complete, the crossover process is activated. Set of trial vectors are created by choosing between the donor vector and target vector. Crossover is done for a set of selected vectors in the population. Number of vectors for crossover is selected based on crossover rate C_R. Trail vectors are generated by using OX operator (Order Crossover) proposed by Davis (1985). A sample illustration of the procedure for this crossover is presented in Section 5.2.4.1.

d) Selection

The selection scheme of DE also differs from that of other evolutionary algorithms (Ali et al., 2009). The population for the next generation is selected from the individual in current population and its corresponding trial vector. Target vector competes with their corresponding trial vector to be selected on to the next generation/iteration. The vector with the better fitness value is copied to the next generation. In this research, vector with minimum assembly line cost is selected. The rule of selection is according to the following rule:

$$x_{i,\mathrm{G+1}} = \begin{cases} \mathrm{Z_{i,G}} & if\ f(Z_{i,G}) < f(x_{i,G}) \\ 0, & otherwise \end{cases} \quad \textbf{(7.3)}$$

The algorithm is terminated if the iteration approaches a predefined criteria, in this case, a predefined maximum number of iterations (generations) is used.

7.1.3 Cost Model Dataset Generation

There are no cost data available to optimize the assembly line cost for a robotic assembly line. This section presents the procedure followed to generate the cost data for the RALB problem. Eight representative precedence graphs and from http://www.assembly-line-balancing.de/, which are widely used in the SALB-I literature (Scholl, 1993) and processing times of robots available in Gao et al. (2009) are used to generate the datasets. The hourly rate of the robots is calculated from the standard procedure of finding annual cost of a capital intensive resource.

$$UAC = IC*(A/P,i,n) \quad \textbf{(7.4)}$$

Here, *UAC* = Equivalent uniform annual cost ($/yr); *i* = annual interest rate and *n*= number of years, *(A/P, i, n)* = capital recovery factor that converts initial cost at year 0 into a series of equivalent uniform annual year-end values.

For given values of *i* and *n, (A/ P, i, n)* can be computed as follows

$$(A/P,i,n) = \frac{i*(i+1)^n}{(i+1)^n - 1} \quad \textbf{(7.5)}$$

Value of (A/ P, i, n) can also be found in interest tables that are widely available. Hourly cost of robot is calculated by dividing the annual cost with total annual hours per year. Cost of robot for a specific time can be calculated with hourly cost of robot. The annual interest rate *i* is assumed as 10% and *n* is assumed as 5 years. Number of annual hours per year is calculated as total working hours multiplied by total number of working days. Number of annual hours is taken as 6000hr/yr (20hr/day*300days/yr). After calculating the cost per hour of a robot, cost of performing a set of task by a robot is calculated by using the performance time. An example is shown for a better understanding on how the cost data is generated.

The steps shows how the cost of a robot for a specific time. Initial robot cost is $1,100,000.

Step 1: Calculate UAC for robot

UAC = *IC (A/P, i, n)*

= 1,100,000* 0.2638)

Uniform Annual Cost = $ 29, 0180

*A/P Value is calculated for 5 years with interest rate 10%

Step 2: Calculate Hourly Rate of the robot

Total number of hours per year = (20 hr/day) (300 day/yr) = 6000 hr/yr.

Cost Per Hour = 290180/6000

= $ 48.36333/hr

*Assembly line is considered to work for 20 hours a day for 300 days in a year.

Step 3: Cost of the robot for a specific time

Time taken to perform a task by robot 1 is 81minutes.

Cost of robot per time = 48.3633*81 /60 = $ **65.2905**

Similarly, cost data for all 32 problems have been generated. It is assumed that costs such as robot cost, setup cost, transportation cost are included in the initial cost of the robot. Table 7.1 is developed based on the UAC cost and subsequent tasks times of robots available. Appendix 4 shows the random robot cost assumed for developing datasets for small size datasets (up to 70 tasks problems) and Appendix 5 for large size datasets (above 89 tasks problems).

7.1.4 Parameter settings

The parameters used in PSO variants are chosen by conducting pilot simulation on three different problems. Different combinations of the parameters are tested until the best combination is achieved. Quality of solution is given importance compared to the computational time in selecting the parameters. For all the four variants of PSO the population size and the total number of generations are kept same. The parameters obtained after the pilot simulation study are presented in Table 7.2

From the experimental studies conducted it is found that proposed DE algorithm works well when the initial size of the population is set to 25 and the mutation factor is set to 0.5 and crossover rate is fixed as 0.9. The total number of generations is fixed to 25. DE parameters chosen are also shown in Table 7.2.

F is a mutation scaling factor of the difference vector (Equation 7.2). This parameter helps to control the evolving rate of the population. F is chosen to be a value in the range [0, 2] for original DE algorithm (Storn and Price, 1997). When small F values are used it could lead to premature convergence and high values can slow down the search (Mohamed et al., 2012). From the literature review it is found that mutation factor 0.5 is better and this value is used for solving all the thirty two problems. Crossover rate (CR) reflects the probability with which the trial individual inherits the actual individual's genes (Feoktistov, 2006). If the CR value is relatively high, this will increase the population diversity and improve the convergence speed (Mohamed et al., 2012). Different levels of crossover rate (0.3, 0.5, 0.7, and 0.9) are tested. Three problems of different task size are evaluated using the different levels of crossover. Best solution is obtained when the CR value is set as 0.9.

Table 7.2 Parameters selected for PSO variants and DE

Parameters for PSO variants	Parameters for DE
Population size: 25	Population size: 25
Number of iterations: 30	Number of iterations: 30
PSO-W: w=0.6, c_1=1 and c_2=2 PSO-C: c1= 2.4 and c2=1.7 PSO-TVIW: w_{max}=0.9, w_{min}=0.4, c_1=1 and c_2=2 PSO-TVAC: c_{1i}=2.5, c_{1f}=0.5, c_{2i}=0.5, c_{2f} =2.5	Mutation factor: 0.5 Crossover rate: 0.9

7.1.5 Performance analysis for straight RALB

Thirty two test problems are solved for the proposed allocation procedure using PSO variants and Differential evolution algorithm. The performances of the model are evaluated to find total assembly line cost in a straight robotic assembly line. The proposed model is coded in C++ and the performances of PSO and DE are tested on Intel core i5 processor (2.3 GHz). The datasets evaluated are divided into two groups: small (up to 70 task problems) and large size datasets (from 89 task problems). Table 7.3 shows the results obtained for the proposed four variants of PSO and DE using consecutive allocation procedure for the objective of minimizing total assembly line cost.

Table 7.3 Results for cost based straight RALB problems using consecutive allocation procedure

Dataset	Total Assembly Line Cost					Dataset	Total Assembly Line Cost				
	PSO-W	PSO-C	PSO-TVIW	PSO-TVAC	DE		PSO-W	PSO-C	PSO-TVIW	PSO-TVAC	DE
25-3	1279	1240	1218	**1218**	**1218**	89-8	3192	3186	3179	3175	**3124**
25-4	992	992	922	**984**	**984**	89-12	2892	2894	2877	2874	**2863**
25-6	830	810	806	804	**803**	89-16	2492	2482	2475	**2471**	2472
25-9	754	748	750	746	**723**	89-21	2314	2312	2310	2304	**2288**
35-4	**945**	**945**	**945**	**945**	**945**	111-9	4307	4245	4271	4244	**4231**
35-5	1356	1340	1339	1326	**1317**	111-13	3404	3364	3360	3360	**3335**
35-7	1347	1335	1332	1322	**1273**	111-17	3396	3392	3364	3317	**3299**
35-12	868	870	861	861	**845**	111-22	2858	2853	2846	2806	**2794**
53-5	2238	2238	2234	**2230**	**2230**	148-10	5681	5657	5625	5621	**5613**
53-7	1859	1857	1846	1783	**1768**	148-14	4270	4250	4232	4230	**4220**
53-10	1683	1673	1677	1675	**1666**	148-21	3900	3895	3884	3764	**3722**
53-14	1379	1366	1354	1334	**1299**	148-29	3822	3816	3815	3796	**3744**
70-7	2378	2372	2375	2329	**2319**	297-19	8512	8500	8434	8412	**8311**
70-10	2276	2275	2262	2263	**2173**	297-29	7879	7812	7807	7725	**7570**
70-14	2013	2007	1997	1970	**1966**	297-38	7796	7795	7791	7738	**7598**
70-19	1794	1798	1775	1758	**1718**	297-50	8568	8561	8345	8337	**8320**

From Table 7.3, it is observed that DE algorithm produces better results for 31 out of 32 datasets when compared with the PSO variants for the allocation done using consecutive allocation procedure for the objective of minimizing the total assembly line cost. When comparing the results of PSO variants and DE, only PSO-TVAC produced results nearer to DE results. The best results found by the proposed algorithms are presented in bold. Average computational time (program running time) taken by four PSO variants and DE algorithm is recorded and is reported here in Table 7.4. Among the four PSO variants and DE, computational time for small size datasets is low for DE algorithm and for large size datasets DE takes more time when compared with other four PSO variants.

Table 7.4 Average Computation Time for consecutive allocation procedure

Problems	Average Computation Time				
	PSO-W	**PSO-C**	**PSO-TVIW**	**PSO-TVAC**	**DE**
25	14	14	14	12	**12**
35	22	23	23	19	**17**
53	33	34	32	29	**27**
70	71	73	67	65	**63**
89	92	94	88	**90**	124
111	142	144	136	**133**	148
148	304	305	**302**	309	352
297	1295	1295	**1292**	1310	1370

7.1.6 Time based and cost based model for straight RALB

Manufacturing industries gives importance for reducing the production cost due to high investment cost incurred for setting up production systems and robotic assembly lines. Industries use robotic assembly lines extensively and these systems are very cost intensive. Due to this, industries need to use the resources available optimally. In the literature survey, no research could be found on optimizing cycle time and total assembly line cost concurrently for RAL. In this research, two models are proposed with dual focus on time and cost to minimize the cycle time and total assembly line cost simultaneously. The first model (cost based model) focusses on the objective of minimizing the total assembly line cost as the primary objective and second model (time based model) focusses on the objective of minimizing cycle time as the primary objective in a straight robotic assembly line.

Results obtained from the previous section for assembly line cost using differential evolution is considered in this section. The consecutive allocation procedure reports the best solution possible for the objective of minimizing the total assembly line cost (cost based model) using the cost data of tasks for DE algorithm. Using the task and robot performance time details the workstation times are calculated. Using the same parameters used for DE, the cycle time (time based model) is also evaluated for all thirty problems for the objective of minimizing cycle time. For the allocation evaluated, the procedure calculates the assembly line cost of each workstation and the total assembly line cost is calculated.

For a sample sequence ***(1-4-5-3-7-9-2-6-8-10-11)***, the allocation is done using the cost based model for the objective of minimizing the total assembly line cost. The workstations are allotted with the tasks and robots are allotted based on the objective of minimizing assembly line cost and using the time data for the problem the workstation times are calculated. Table 7.5 shows the task and robot allocation for the sample sequence. Figure 7.4 shows the workstation times and assembly line cost of each workstation.

Table 7.5 Task and robot allocation using cost based model

Workstation	Tasks	Robot Allotted
Workstation 1	1, 4, 5	Robot 2
Workstation 2	3, 7, 9	Robot 3
Workstation 3	2, 6	Robot 4
Workstation 4	8, 10, 11	Robot 2

The workstation time is calculated using the time data available in Table 5.2. Time at Workstation 1 (Robot 2) =37+41+36=114, Time at Workstation 2 (Robot 3) = 38+40+41=119, Time at Workstation 3 (Robot 4) =42+71=113 and Time at Workstation 4 (Robot 3) = 42+46+38=126. The cycle time is 126 and the total assembly line cost is 429.

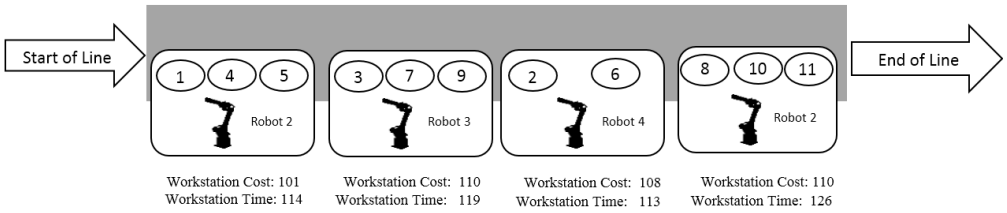

Figure 7.4 Workstation cost and cycle time allocation done using cost based model

The same sequence is considered for the allocation of tasks and robots with an objective of minimizing cycle time using the time data of the tasks. Consecutive procedure explained in Section 5.1.1 is used for the cycle time calculation. Table 7.6 shows the allocation of tasks and robots allotted using the time based model.

Using Table 7.1, the cost of the assembly at each workstation is calculated for the allocation made based on the objective of minimizing cycle time and the overall assembly line cost is calculated by taking the sum of the cost to perform the tasks at

each workstation using the cost data. Cost for Workstation 1 (Robot 2) =33+36+32=101, Cost for Workstation 2 (Robot 3) = 35+37+38=110, Cost for Workstation 3 (Robot 4) =40+68+42=150 and Cost for Workstation 4 (Robot 3) = 40+33=73. Figure 7.5 shows the robot and task allocation with the workstation cost and workstation time in a straight robotic assembly line. The cycle time is 157 and the total assembly line cost is 434.

Table 7.6 Task and robot allocation using time based model

Workstation	Tasks	Robot Allotted
Workstation 1	1, 4, 5	Robot 2
Workstation 2	3, 7, 9	Robot 3
Workstation 3	2, 6, 8	Robot 4
Workstation 4	10, 11	Robot 2

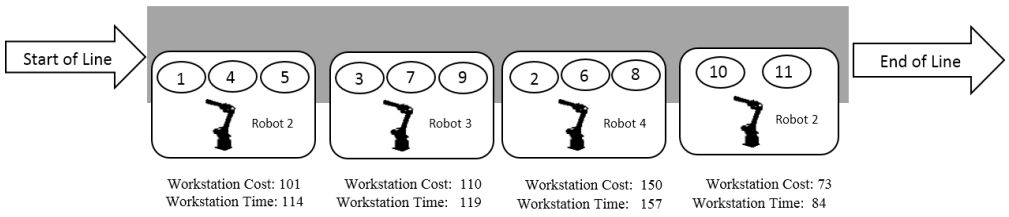

Figure 7.5 Workstation cost and cycle time using time based model

Results of thirty two problems generated are compared for both the objectives in a straight robotic assembly line. The datasets are divided into two groups: small size datasets and large size datasets. The complete details of the results obtained by using the time based and cost based model for small size datasets (Problem No: 1 to 16) and for large size datasets (Problem No: 17 to 32) are presented in Table 7.7. From the tables it is evident that cost based model is better in terms of minimizing the total assembly line cost when compared with time based model for both the groups of datasets and cycle time is better for time based data model when compared with the cost based data model.

Assembly line cost evaluated using cost based model is lower when compared to assembly line cost obtained for time based model in a straight robotic assembly line. Differences in assembly line cost between two models are taken and the average of the difference is taken to calculate the average cost savings. It is observed the average cost saving by using cost based model for small size datasets is around 205 cost units. The

average cost which can be saved by using cost based model over time based model for large size dataset is found to be 573 cost units. Figure 7.6 and Figure 7.7 represents the difference in cost of the assembly line (cost saving potential) between the time based model and cost based model and the graph represents the cost saved by using cost based model when compared to the time based model.

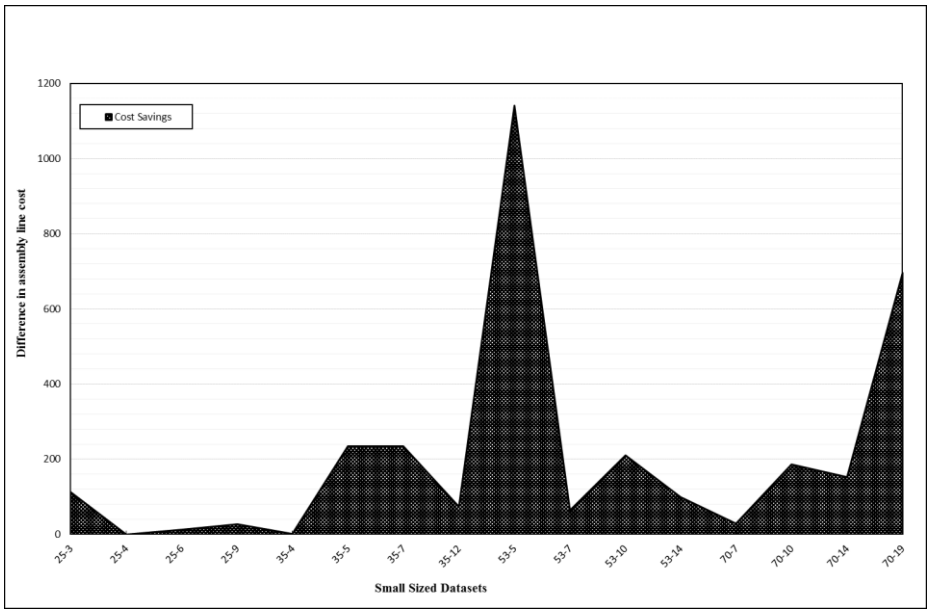

Figure 7.6 Cost saving potential in small size datasets for cost based model in straight RALB

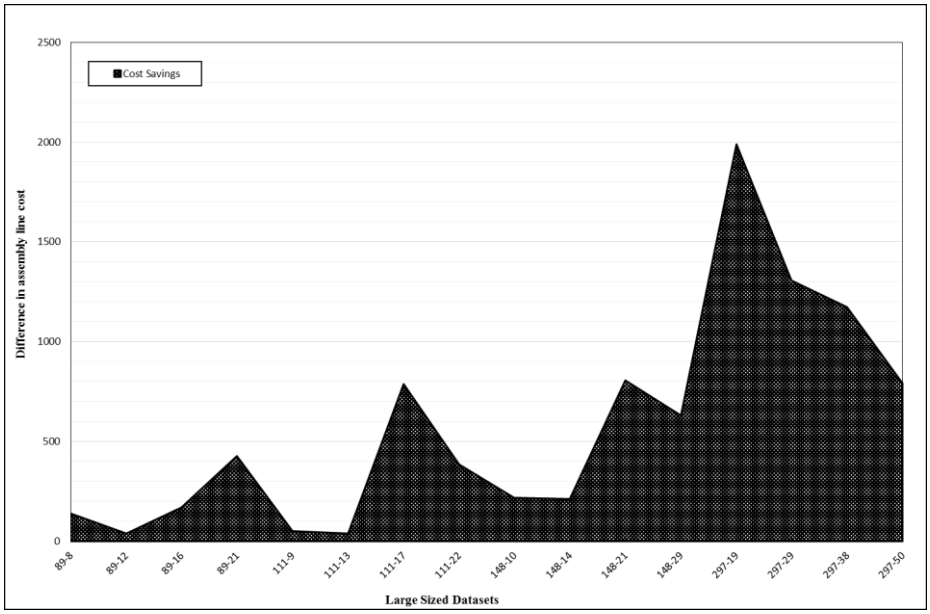

Figure 7.7 Cost saving potential in large size datasets for cost based model in straight RALB

Table 7.7 Comparison of assembly line cost and cycle time for two models in straight RALB

Problem No:	Problem Dataset	Assembly Line Cost		Cycle Time		Problem No:	Problem Dataset	Assembly Line Cost		Cycle Time	
		Cost Model	Time Model	Cost Model	Time Model			Cost Model	Time Model	Cost Model	Time Model
1	25-3	**1218**	1331	706	**503**	17	89-8	**3124**	3264	516	**461**
2	25-4	**984**	984	299	**293**	18	89-12	**2863**	2904	383	**320**
3	25-6	**803**	815	221	**200**	19	89-16	**2472**	2641	292	**219**
4	25-9	**723**	750	124	**114**	20	89-21	**2288**	2716	244	**170**
5	35-4	**945**	947	374	**342**	21	111-9	**4231**	4284	698	**521**
6	35-5	**1317**	1551	464	**333**	22	111-13	**3335**	3375	438	**321**
7	35-7	**1273**	1507	279	**211**	23	111-17	**3299**	4088	349	**243**
8	35-12	**845**	918	130	**104**	24	111-22	**2794**	3179	293	**184**
9	53-5	**2230**	3371	561	**449**	25	148-10	**5613**	5832	881	**586**
10	53-7	**1768**	1832	362	**295**	26	148-14	**4220**	4431	561	**419**
11	53-10	**1666**	1877	252	**224**	27	148-21	**3722**	4528	321	**273**
12	53-14	**1299**	1398	168	**142**	28	148-29	**3744**	4374	236	**190**
13	70-7	**2319**	2348	504	**430**	29	297-19	**8311**	10301	675	**594**
14	70-10	**2173**	2360	351	**262**	30	297-29	**7570**	8876	503	**394**
15	70-14	**1966**	2118	247	**194**	31	297-38	**7598**	8771	365	**305**
16	70-19	**1718**	2413	176	**139**	32	297-50	**8320**	9112	331	**221**

When comparing the cycle time among two models it is observed that cycle time obtained by time based model is lower. The average reduction in cycle time by using time based model for small size datasets is 61 cycle time units and for large size datasets average reduction in cycle time is 104 units.

Depending upon the priority of the management, the primary focus between time and cost could vary at different time horizon. The appropriate model could be selected based on the priority of the management.

7.2 U-shaped RALB – Assembly line cost

This section presents the procedure followed to find out the total assembly line cost in a U-shaped robotic assembly line. Consecutive allocation procedure is proposed for task and robot allocation with an objective of minimizing the total assembly line cost in a U-shaped robotic assembly line.

7.2.1 Task and robot allocation procedure in U-shaped RALB

This procedure is used to calculate the total assembly line cost of a U-shaped robotic assembly line. The procedure for task allocation varies from the straight robotic assembly line. U-shaped allocation allows more possibilities for task allocation. Tasks are allocated to the workstation by moving forward and backward through the precedence diagram in contrast to a typical forward move in the traditional assembly systems. An initial assembly line cost (P_0) is calculated to start the procedure. The procedure tries to allocate the maximum number of tasks to the workstations without violating the precedence constraints. If the initial P_0 cannot accommodate all the tasks, P_0 is incremented by one and the procedure is repeated to accommodate all the tasks. The allocation done gives the cost to perform the task allotted to each workstation. The total assembly line cost is calculated by taking the sum of cost incurred at each workstation. An illustration is provided in this section which explains the task and robot allocation and calculation of energy consumption in a U-shaped robotic assembly line. Sequence of tasks which meets the precedence constraints is considered for illustration. Let, the sequence be, ***(1-4-5-3-7-9-2-6-8-10-11)***: 11 task and 4 workstation problem is considered for the illustration. Performance cost data details of each tasks and robots are presented in Table 7.1.

Step 1. Using Equation 7.1, P_0 is calculated and it is found to be 98.

Step 2. For the initial P_0, the procedure tries to allocate the tasks to the workstations starting from the first workstation. Procedure checks the both sides of the sequence if any of the robots could perform the tasks within P_0. Different possible combinations are available in U-shaped. The procedure chooses the combination which minimizes the cost at each workstation.

Step 3. Next workstation is open and remaining tasks from the sequence are allocated if the initial assembly line cost cannot allocate all the tasks.

Step 4. The initial value of assembly line cost is incremented if tasks are still left unassigned for the initial value and Step 2 and 3 are repeated until all tasks get assigned to the workstation.

Step 5. Robots are allotted to each workstation with certain set of tasks. Robots which perform the allotted tasks in minimum cost is allotted to the workstation

Step 6. The sum of cost of each workstation gives the total assembly line cost.

For cost based model in U-shaped robotic assembly, when the allocation was attempted with initial P_0 it was found that tasks 2, 7 and 9 are left unassigned. P_0 is incremented till 108 to accommodate all the tasks to the four workstations. The total assembly line cost of the given sequence is calculated as 419 cost units. Figure 7.8 shows the allocation based on the cost based model in a U-shaped RAL.

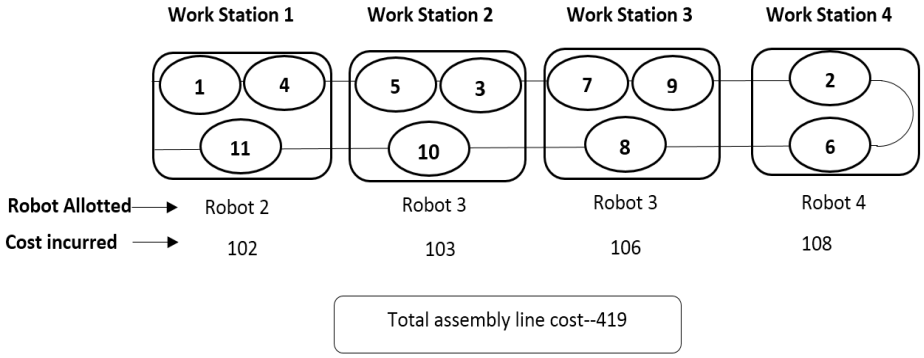

Figure 7.8 Final task and robot allocation in a U-shaped RALB for cost

7.2.2 PSO Variants and DE to solve cost based model in U-shaped RALB

Since this problem also falls under the category of NP-hard, four variants of particle swarm optimization algorithm and Differential Evolution (DE) algorithms are proposed to solve both the proposed models. Four variants as explained in Section 5.2 are implemented to solve the problem. Details of Differential Evolution algorithm developed to solve the problem are presented in Section 7.1.2. The same set of parameters used for solving cost based model in straight line is adopted to solve U-shaped cost based robotic assembly line problems.

7.2.3 Performance analysis for cost based U-shaped RALB

Thirty two test problems are solved using PSO variants and Differential evolution algorithm for the objective of minimizing the assembly line cost. The proposed model is coded in C++ and the performances are tested on Intel core i5 processor (2.3 GHz). The datasets evaluated are divided into two groups: small size datasets and large size datasets. Table 7.8 shows the results obtained for the proposed four variants of PSO and DE for U-shaped robotic assembly line balancing problems. The results reported are the best solution found using the four variants of PSO and DE From the table it is analyzed that the results obtained for DE reports better results when compared with the

results obtained using the four variants. There is significant improvement in the solution quality when DE algorithm is used to generate the results.

Average computational time (program running time) taken by 4 PSO variants and DE algorithm is recorded and is reported here in Table 7.9. Computational time to perform the datasets with small size is low whereas for large size datasets computational time is high. Among the four variants and DE, computational time for small size datasets is low for PSO-TVAC algorithm and for large size datasets DE takes lesser time when compared with the 4 PSO variants. Among the four variants the time taken by PSO-TVAC is lower for all the thirty two datasets. Results obtained using DE show that DE is better for all the problems in terms of the objective, its robustness and computational efficiency can be improved by fine tuning the parameters for getting the solution at a faster rate for the small size datasets.

Table 7.8 Results for cost based U-shaped RALB problems using PSO variants and DE

Dataset	Total Assembly Line Cost					Dataset	Total Assembly Line Cost				
	PSO-W	PSO-C	PSO-TVIW	PSO-TVAC	DE		PSO-W	PSO-C	PSO-TVIW	PSO-TVAC	DE
25-3	1229	1229	1222	1225	**1206**	89-8	3262	3146	3152	3126	**3121**
25-4	1010	991	987	984	**965**	89-12	2865	2850	2831	2792	**2773**
25-6	822	815	798	796	**778**	89-16	2470	2448	2413	2392	**2388**
25-9	730	720	715	719	**704**	89-21	2312	2300	2286	2279	**2254**
35-4	945	945	945	**945**	**945**	111-9	4209	4242	4199	4161	**4135**
35-5	1422	1337	1326	1319	**1299**	111-13	3329	3322	3308	3302	**3294**
35-7	1334	1312	1308	**1306**	**1306**	111-17	3292	3279	3255	3237	**3209**
35-12	857	858	855	806	**795**	111-22	2821	2789	2762	2728	**2730**
53-5	2238	2195	2159	**2167**	2195	148-10	5596	5585	5526	5509	**5488**
53-7	1747	1739	1730	**1720**	1739	148-14	4279	4167	4169	4166	**4164**
53-10	1683	1678	1661	1652	**1649**	148-21	3768	3722	3740	3690	**3664**
53-14	1313	1309	1306	1294	**1266**	148-29	3690	3678	3595	3586	**3574**
70-7	2378	2368	2367	2346	**2339**	297-19	8510	8391	8340	8340	**8253**
70-10	2237	2223	2208	2185	**2152**	297-29	7707	7650	7638	7676	**7460**
70-14	1980	1954	1947	1929	**1918**	297-38	7627	7622	7542	7551	**7514**
70-19	1700	1733	1702	1689	**1659**	297-50	8364	8378	8277	**8234**	**8234**

Table 7.9 Average Computational time for cost based U-shaped RALB

Problems	Average Computation Time				
	PSO-W	**PSO-C**	**PSO-TVIW**	**PSO-TVAC**	**DE**
25	18	17	18	**16**	17
35	24	25	25	**22**	24
53	36	39	42	**32**	33
70	75	78	77	**69**	72
89	98	99	98	95	**93**
111	148	145	149	139	**135**
148	317	315	320	314	**310**
297	1350	1355	1362	1345	**1340**

7.2.4 Time and Cost based model in U-shaped RALB

Resources at an industry need to be optimally used to reduce the loss incurred by operating cost intensive robotic assembly line systems. From the literature, it is found that no research is found where both the assembly line cost and cycle time are optimized concurrently in a U-shaped robotic assembly line. In this research, two models are proposed with dual focus on time and cost to minimize the cycle time and total assembly line cost simultaneously. The first model (cost based model) focusses on the objective of minimizing the total assembly line cost as the primary objective and second model (time based model) focusses on the objective of minimizing cycle time as the primary objective in a straight robotic assembly line (RAL).

In the previous section, DE algorithm reported the best solution among the proposed algorithms. The results obtained from DE are used in this section for calculating the cycle time for the cost based model. Using the task and robot details, the workstation times are calculated and the workstation time which is the maximum is the cycle time of the allocation.

A sample sequence which is used in the previous section to show the allocation based on the objective of minimizing the total assembly line cost is used here. Table 5.2 is used for finding the workstation time. Time at Workstation 1 (Robot 2) =37+41+38=116, Time at Workstation 2 (Robot 3) = 38+33+41=112, Time at Workstation 3 (Robot 3) =40+34+31=115 and Time at Workstation 4 (Robot 4) = 42+71=113. The cycle time of the U-shaped robotic assembly line is 116 and the total assembly line cost is 419 as shown in Figure 7.9.

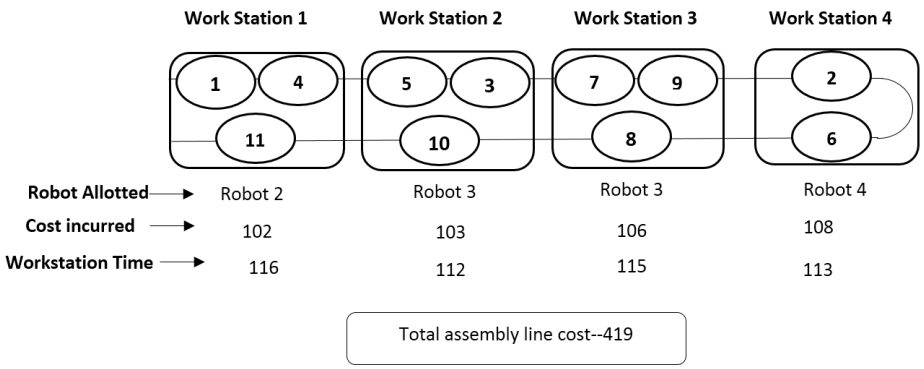

Figure 7.9 Workstation cost and time in U-shaped RALB using cost based model

Using the same set of parameters used to solve the cost based model, the allocation procedure used for U-shaped allocation in robotic assembly line for the objective of minimizing the cycle time (time based model) is evaluated (Table 5.2). Procedure explained in Section 5.5.3 is used for allocation of tasks and robots based on the objective of minimizing the cycle time. For the allocation made using this model, the assembly line cost is calculated. Figure 7.10 shows the final allocation of tasks and robots based on the objective of minimizing the cycle time. Using Table 7.1, the overall assembly line cost is calculated by taking the sum of the cost to perform the tasks at each workstation. Cost for Workstation 1(Robot 2) =33+36+33=102, Cost for Workstation 2(Robot 3) = 30+31+38=99, Cost for Workstation 3(Robot 4) =33+76=150 and Cost for Workstation 4(Robot 3) = 40+47+31=118. The cycle time is 124 and the total assembly line cost is 430 when the allocation is done based on the objective of minimizing the cycle time in a U-shaped robotic assembly line.

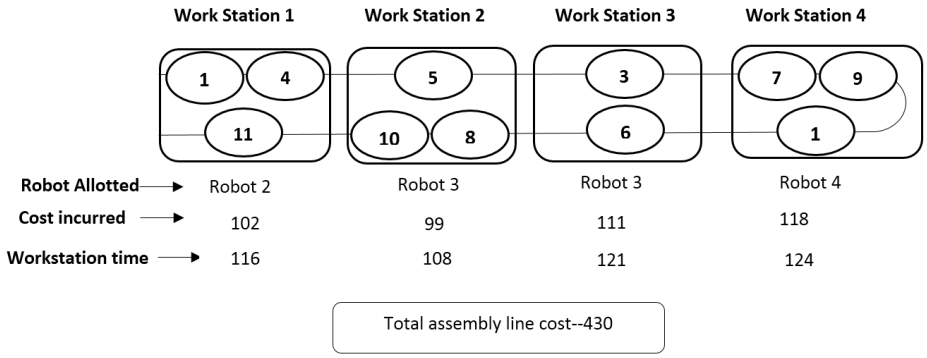

Figure 7.10 Workstation cost and time in U-shaped RALB using time based model

Both the objectives of minimizing cycle time and assembly line cost in a U-shaped robotic assembly line are tested on the thirty two problems generated. Two groups of datasets (small and large size datasets) are available. The detailed results obtained by using the time based and cost based model for U-shaped robotic assembly line for small size datasets (Problem No: 1 to 16) and for large size datasets (Problem No: 17 to 32) are presented in Table 7.10 . From the tables it is evident that cost based model is better in terms of minimizing the total assembly line cost when compared with time based model for both the groups of datasets and cycle time is better for time based data model when compared with the cost based data model for U-shaped robotic assembly line except for two datasets (53-7 and 148-14).

Assembly line cost evaluated using cost based model is lower when compared to assembly line cost obtained for time based model in a U-shaped robotic assembly line. Difference in assembly line cost between two models is taken and the average of the difference is taken to calculate the average cost savings. It is observed the average cost saving by using cost based model for small size datasets is around 237 cost units. The average cost which can be saved by using cost based model over time based model for large dataset is found to be 502 cost units.

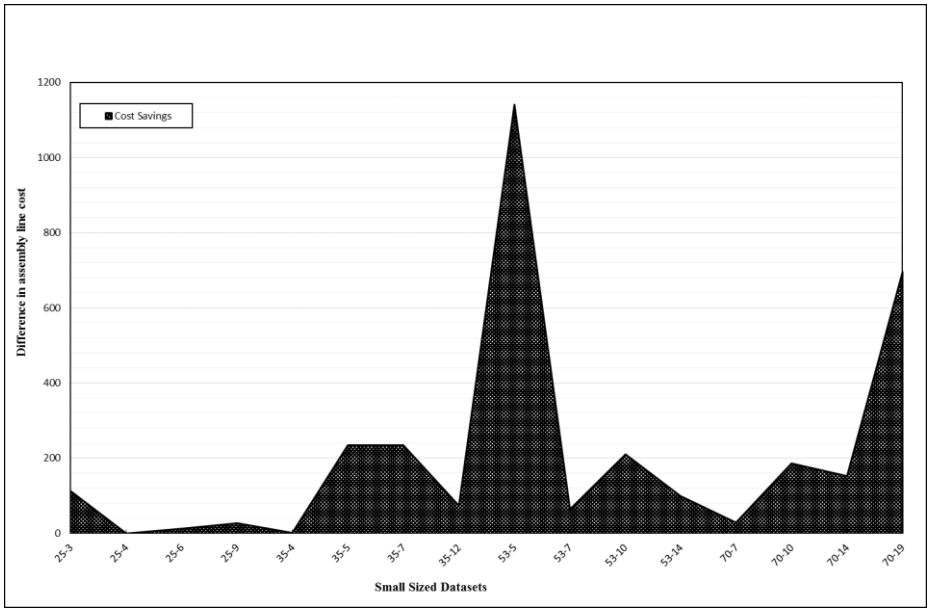

Figure 7.11 Cost saving potential in small size datasets for cost based model in U-shaped RALB

Table 7.10 Comparison of assembly line cost and cycle time for two models in U-shaped RALB

Problem No:	Problem Dataset	Assembly Line Cost		Cycle Time		Problem No:	Problem Dataset	Assembly Line Cost		Cycle Time	
		Cost Model	Time Model	Cost Model	Time Model			Cost Model	Time Model	Cost Model	Time Model
1	25-3	**1206**	1451	583	**500**	17	89-8	**3121**	3174	598	**481**
2	25-4	**965**	989	**303**	318	18	89-12	**2850**	2921	425	**319**
3	25-6	**778**	1101	189	**183**	19	89-16	**2448**	2513	252	**219**
4	25-9	**704**	740	114	**110**	20	89-21	**2254**	2561	216	**170**
5	35-4	**945**	947	355	**343**	21	111-9	**4135**	4343	690	**522**
6	35-5	**1299**	1582	473	**336**	22	111-13	**3294**	3300	366	**319**
7	35-7	**1306**	1439	268	**212**	23	111-17	**3209**	3809	311	**242**
8	35-12	**795**	907	128	**103**	24	111-22	**2730**	3049	238	**181**
9	53-5	**2195**	3512	660	**447**	25	148-10	**5596**	5697	818	**619**
10	53-7	1739	**1725**	359	**283**	26	148-14	4164	**4161**	446	**411**
11	53-10	**1649**	1921	253	**220**	27	148-21	**3664**	4438	321	**270**
12	53-14	**1266**	1295	162	**144**	28	148-29	**3574**	5003	230	**188**
13	70-7	**2339**	2439	483	**427**	29	297-19	**8253**	8913	686	**591**
14	70-10	**2152**	2263	339	**264**	30	297-29	**7460**	8702	513	**390**
15	70-14	**1918**	2089	217	**195**	31	297-38	**7514**	8579	357	**292**
16	70-19	**1659**	2322	168	**138**	32	297-50	**8234**	9373	305	**222**

Figure 7.11 and Figure 7.12 represents the difference in cost of the assembly line (cost saving potential) between the time based model and cost based model and the graph represents the cost saved by using cost based model when compared to the time based model.

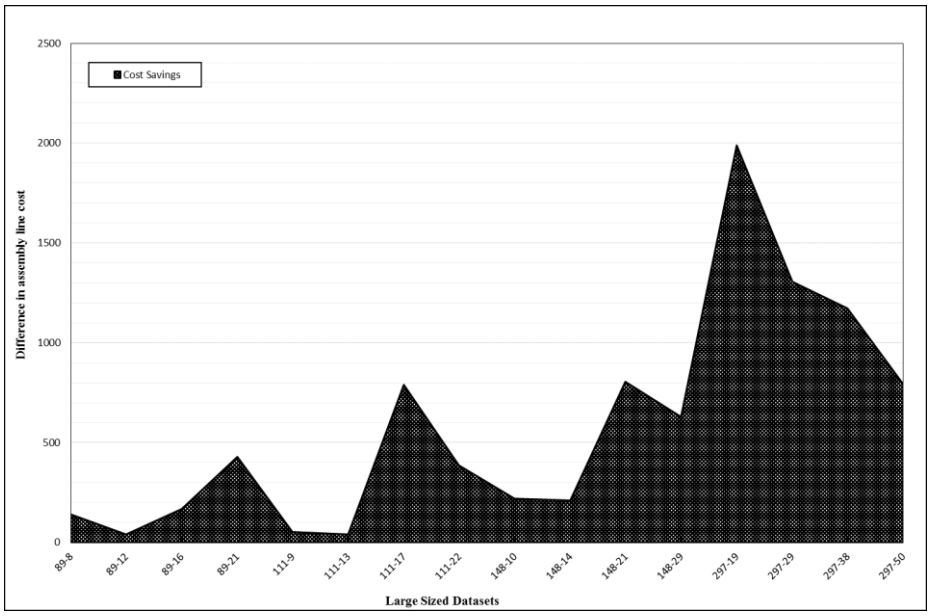

Figure 7.12 Cost saving potential in large size datasets for cost based model in U-shaped RALB

When comparing the cycle time among two models it is observed that cycle time obtained by time based model is lower. The average reduction in cycle time by using time based model for small size datasets is 52 cycle time units and for large size datasets average reduction in cycle time is 84 units.

7.3 Comparison of straight and U-shaped RALB

The total assembly line cost and cycle time obtained using cost based model for straight and U-shaped robotic assembly line are compared. Table 7.11 is formed by extracting the results from Table 7.7 and Table 7.10 obtained for minimizing total assembly line cost from straight and U-shaped robotic assembly line using cost based model results.

Table 7.11 Comparison of assembly line cost - straight and U-shaped RALB

Problem No:	Problem Dataset	Assembly Line Cost		Problem No:	Problem Dataset	Assembly Line Cost	
		Straight RALB	U-Shaped RALB			Straight RALB	U-Shaped RALB
1	25-3	1218	**1206**	17	89-8	3124	**3121**
2	25-4	984	**965**	18	89-12	2863	**2850**
3	25-6	803	**778**	19	89-16	2472	**2448**
4	25-9	723	**704**	20	89-21	2288	**2254**
5	35-4	**945**	**945**	21	111-9	4231	**4135**
6	35-5	1317	**1299**	22	111-13	3335	**3294**
7	35-7	**1273**	1306	23	111-17	3299	**3209**
8	35-12	845	**795**	24	111-22	2794	**2730**
9	53-5	2230	**2195**	25	148-10	5613	**5596**
10	53-7	1768	**1739**	26	148-14	4220	**4164**
11	53-10	1666	**1649**	27	148-21	3722	**3664**
12	53-14	1299	**1266**	28	148-29	3744	**3574**
13	70-7	2319	2339	29	297-19	8311	**8253**
14	70-10	2173	**2152**	30	297-29	7570	**7460**
15	70-14	1966	**1918**	31	297-38	7598	**7514**
16	70-19	1718	**1659**	32	297-50	8320	**8234**
Average Cost Savings for U-shaped		**21 cost units**		**Average Cost Savings for U-shaped**		**63 cost units**	

The results indicate that total assembly line cost is very low for U-shaped robotic assembly line when compared to the total assembly line cost in straight robotic

assembly line. Thirty out of thirty two datasets yielded lower assembly line cost for U-shaped robotic assembly line. Average cost savings which is achieved by using U-shaped robotic assembly line over straight robotic assembly line is calculated as 21 cost units for small size datasets (up to problem number 16) and 63 cost units for large size datasets. Cycle time of both straight and U-shaped robotic assembly line obtained using time based model are extracted from Table 7.7 and Table 7.10 and the results are presented in Table 7.12.

Table 7.12 Comparison of cycle time - straight and U-shaped RALB

Problem No:	Problem Dataset	Cycle Time		Problem No:	Problem Dataset	Cycle Time	
		Straight RALB	U-Shaped RALB			Straight RALB	U-Shaped RALB
1	25-3	503	**500**	17	89-8	461	**481**
2	25-4	**293**	318	18	89-12	320	**319**
3	25-6	200	**183**	19	89-16	**219**	**219**
4	25-9	114	**110**	20	89-21	170	**170**
5	35-4	**342**	343	21	111-9	521	**521**
6	35-5	**333**	336	22	111-13	321	**319**
7	35-7	**211**	212	23	111-17	243	**242**
8	35-12	104	**103**	24	111-22	184	**181**
9	53-5	449	**447**	25	148-10	586	619
10	53-7	295	**283**	26	148-14	419	**411**
11	53-10	224	**220**	27	148-21	273	**270**
12	53-14	**142**	144	28	148-29	190	**188**
13	70-7	430	**427**	29	297-19	594	**591**
14	70-10	**262**	264	30	297-29	394	**390**
15	70-14	**194**	195	31	297-38	305	**292**
16	70-19	139	**138**	32	297-50	**221**	222
Average % Reduction in cycle time using U-shaped			**0.5%**	**Average % Reduction in cycle time using U-shaped**			**0.15%**

From Table 7.12, it is observed that cycle time of U-shaped robotic assembly line obtained using the time based model is lower than the cycle time for straight robotic assembly line problems for 28 out of 32 problems. The average percentage reduction in cycle time by U-shaped layout for the small size datasets is computed as 1.5% and the average percentage reduction in cycle time for large size datasets is computed as 0.5%. Twenty three out of thirty two problems obtained better cycle time for U-shaped robotic assembly line when compared with straight robotic assembly line. Hence it is

concluded from this study that U-shaped robotic assembly line performs better than straight robotic assembly line for the objective of minimizing cycle time as well as minimizing total assembly line cost.

7.4 Summary

In this chapter, a new robotic assembly line balancing problem with an objective of minimizing assembly line cost in addition to cycle time is developed. The author could not find any literature on optimizing assembly line cost in robotic assembly line systems to date. The work presented in this chapter is an important addition to the literature where majority of the work on robotic assembly line dealt with the objective of minimizing cycle time. In a large assembly line, different robotic systems can be used to perform the tasks in the assembly line. Robot needs to be assigned to the work stations based on the minimum cost required to perform the tasks which are allocated to the work stations based on heuristic methods. Consecutive allocation is implemented for robot and task allocation for straight and U-shaped robotic assembly line. Robots are allotted to the work station based on the cost and the total cost of the assembly line is calculated. Since this problem is well known as NP-hard, PSO variants and differential evolution algorithm has been proposed to solve this problem. Data sets have been developed and tested with the proposed algorithms and the results are reported for both straight and U-shaped RALB problems. Among the PSO variants and DE, it is observed that DE reports better quality of solution. Parametric study is conducted on selected problems to choose the efficient set of parameters for PSO and DE. Computational time is also reported.

As part of the experimental study, the cost of the assembly line is also calculated for the allocation done based on the objective of minimizing cycle time. The potential cost savings which is achieved by allocating tasks and robots based on the cost based model is calculated for both straight and U-shaped robotic assembly line. Cycle time for both the layouts is also calculated for the allocation done based on cost. Assembly line cost and cycle time obtained for both the layouts are compared and it is concluded from the results obtained that U-shaped robotic assembly line reports better cycle time and assembly line cost when compared with straight robotic assembly line. These models can be strongly recommended to solve problem instances that occur in practice, regardless of the characteristics of the actual real-world problem.

CHAPTER 8

Particle Swarm Optimization & Differential Evolution for RALB Problem to Maximize Line Efficiency

Efficiency is a crucial factor in industrial production lines as it results in an improved production and checks the utilization of available resources. When the line efficiency is high, it can be inferred that all resources are well utilized. Thus, excess resources can be allocated for performing other tasks. Industries spend appropriate cost of production when the line efficiency is high. Industries would be able to produce more output in shorter time and will be meeting the demand of the customers if the efficiency is high. Maximizing line efficiency is the key to profitability. Two models are proposed to calculate line efficiency in a robotic assembly line. The first model is based on the line efficiency calculated based on the workstation times which are calculated using time data of the tasks. Second model is developed based on the objective of minimizing energy consumption. The workstation times are calculated for the allocation made based on this objective. The workstation times are used to calculate the line efficiency. Metaheuristics like PSO and DE are used to solve RALB problem with an objective of maximizing the line efficiency of two layouts (straight and U-shaped) of robotic assembly line. This chapter presents the procedure proposed and the results obtained using the metaheuristics.

8.1 Line Efficiency calculation in Straight and U-shaped RALB

This chapter describes a type of robotic assembly line balancing problem, in which assembly tasks are allocated to the workstation and the workstation is allotted with a best fit robot which performs the allotted tasks in minimum time with an objective of maximizing line efficiency. The objective function evaluated in this research for straight and U-shaped type of assembly line is Line Efficiency (LE). Line efficiency is calculated when the workstation times are minimized in a robotic assembly line. Cycle time and smoothness index of the efficient assembly line is also calculated. Two layout types of robotic assembly line are evaluated. Particle swarm optimization (PSO) and Differential evolution (DE) are the two metaheuristic algorithms used as the

optimization tool to solve this problem. Benchmark datasets are used for solving the two layouts (straight and U-shaped).

Steps involved in Line Efficiency Calculation

i. Allocation of tasks to the workstations based on the consecutive procedure (Section 5.1.1). This procedure tries to allocate tasks to the workstation by minimizing the workstation time.
ii. Workstations are allotted with the robots which perform the allotted tasks in minimum time.
iii. The workstation time is calculated for the allocation of tasks and robots.
iv. The workstation with the maximum workstation time is the cycle time of the allocation.
v. Line efficiency is calculated using Equation 8.1. Line efficiency is the direct indication of the efficiency of a given assembly line. The efficiency gives results in percentage from 0 to 100%.

$$LE = \frac{\sum_{k=1}^{N_w} S_k}{N_w * c} * 100 \qquad \textbf{(8.1)}$$

Here S_k is the workstation time, N_w is total number of workstations and C is the cycle time

Figure 8.1 shows the tasks allocation and workstation times calculated using the consecutive allocation procedure for a straight robotic assembly line. In the assembly line sequence considered number of workstation is 4 and cycle time is 143.

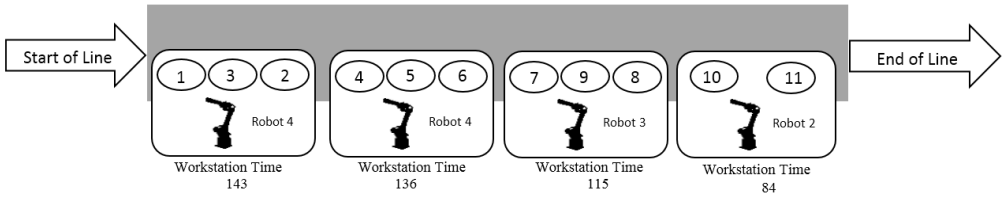

Figure 8.1 Task allocation and Workstation times in straight RALB

Straight line robotic assembly line's efficiency is calculated as follows:

$$LE = \frac{\sum_{k=1}^{N_w} S_k}{N_w * c} * 100 = (143+136+115+84)/ (4*143)*100 = 83.56\%$$

Task and robot allocation in a U-shaped assembly line is done using the same procedure except for the task allocation, the tasks are selected from both the sides of the sequence and constraints are checked (Refer Section 5.5.3). Figure 8.2 shows the tasks allocated for U-shaped robotic assembly line for the same sequence used for straight robotic assembly line. The line efficiency is calculated using the Equation 8.1. U-shaped robotic assembly line's efficiency is: *LE*= (121+115+107+116)/ (4*121)*100=94.83%.

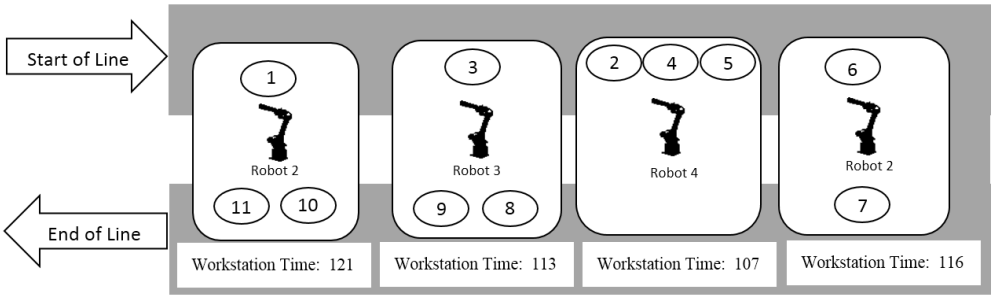

Figure 8.2 Allocation in a U-shaped RALB and workstation times

8.2 PSO and DE to solve time based model line efficiency

Since the assembly line balancing problems falls under the category of NP-hard, two different metaheuristics are used to solve the RALB problem with an objective of maximizing the line efficiency when the workstation times are minimized. Two layout of RALB: straight and U-shaped RAL are solved using these two metaheuristics.

8.2.1 Particle Swarm Optimization

PSO algorithm is proposed to solve the RALB problem with an objective of maximizing the line efficiency. PSO algorithm starts with an initial population and initial velocity. Heuristics explained in Section 5.1.1 are used for generating the initial population. Velocity pairs used in Section 5.1.1 is adopted for the velocity update. When the local best and global best are selected, the particle with higher assembly line

efficiency is considered. PSO is used to solve both straight and U-shaped layouts of robotic assembly line.

8.2.2 Differential Evolution

Differential Evolution algorithm is also proposed to solve the RALB problem with an objective of maximizing the line efficiency. Same procedure used for generating the initial population in PSO is adopted for DE also. DE procedure undergoes mutation, cross and selection operation. Section 7.1.2 explains the details on how to implement DE. Same procedure is adopted for solving this model of RALB also. The difference in DE model will be in the selection operation, where the fitness value of target and trial vectors is compared and the vector which has a better fitness is copied to the next generation. In this model, the fitness value compared is the line efficiency. The vector with higher efficiency is copied to the next generation. Two layouts (straight and U-shaped) of RAL are solved using Differential evolution.

8.2.3 Parameters for PSO and DE

Performance of PSO and DE mainly relies on the parameters selected. Parameters are selected based on the tests conducted in order to get a satisfactory solution quality in an acceptable time span. Influence of each parameter on the solution quality is tested. Three datasets of different task size are chosen to find the best combination of parameters. Three parameters fine-tuned for PSO are stopping condition, population size and acceleration coefficients. For DE, the parameters fine-tuned are population size, stopping condition, cross over rate. Table 8.1 summarizes the parameters used to solve RALB problems with an objective of maximizing line efficiency when the workstation times are minimized.

Table 8.1 Parameters for PSO and DE for RALB problem

Parameters for PSO	Parameters for DE
Population size: 25	Population size: 25
Number of iterations: 30	Number of iterations: 30
Acceleration Coefficients: c1=1, c2=2 and c3=2	Mutation factor: 0.5 Crossover rate: 0.9

8.2.4 Performance analysis of PSO and DE for straight RALB

All the 32 test problems are evaluated using the proposed PSO and DE algorithm. The non-deterministic nature of the algorithm and problem makes it necessary to run

same problem multiple times. Each problem is run ten times and most of the runs converged to the same solution for each of the problems. The results obtained by evaluating 32 test problems are presented in Table 8.2. Column I presents the problem evaluated, Column II, III and IV presents the line efficiency, cycle time and smoothness index of the best solution evaluated using PSO algorithm. Colum V, VI and VII presents the results obtained using DE algorithm. Results reported are the best solution obtained for both the algorithms. Smoothness index of the assembly line is calculated for the allocation obtained. Smoothness Index is an index to indicate the relative smoothness of a given assembly line. When smoothness index is zero it indicates that the assembly line is perfectly balanced. Equation for smoothness index (SI) is:

$$SI = \sqrt{\frac{\sum_{k=1}^{m} (S_{\max} - S_k)^2}{N_w}} \qquad \textbf{(8.2)}$$

Here S_{max} is the station with the maximum station time and S_k is the current workstation time, k is the workstation number (being evaluated) and N_w is the number of workstations.

The problems are classified into two categories based on the tasks size: small (up to 70 tasks problems) and large (above 89 tasks problems). Proposed algorithms allocate robots to the workstation by allocating the robots and tasks which minimizes the workstation times. Using the solution obtained, the workstation time and cycle time of the assembly line are calculated. The workstation time and cycle time is used for calculating the line efficiency of the assembly line. Thus line efficiency is obtained by minimizing the workstation times. From the experiments conducted it is observed that 27 out of 32 datasets could obtain better line efficiency for PSO when compared to DE. The efficiency reported for 25-3 problem is same for both the algorithms. PSO obtained average improvement of 4.2% in the line efficiency when compared to PSO for small size datasets (up to 70 task problems) and average improvement in the efficiency line efficiency for large size datasets (above 89 task problems) when using PSO is found to be 1.0%. Percentage could be low for large size datasets because both PSO and DE reach solutions which are nearer to each other. However, the cycle time of PSO when compared with cycle time of DE is lower for the five datasets which did not yield better efficiency. When comparing the smoothness index between two algorithms, it is

observed that PSO produces better smoothness index when compared to DE. Table 8.3 reports the computational time for both the algorithms.

Table 8.2 Line Efficiency of straight RALB for PSO and DE

Work Station/ Robots	PSO			DE		
	Line Efficiency %)	Cycle Time	Smoothness Index	Line Efficiency (%)	Cycle Time	Smoothness Index
25-3	**97.3**	503	18.6	**97.3**	503	18.6
25-4	**97.1**	294	13.4	88.6	329	40.4
25-6	**90.5**	200	22.5	88.2	208	29.3
25-9	**87.4**	110	20.4	84.5	114	23.3
35-4	**99.4**	342	2.5	98.0	347	9.6
35-5	**95.2**	329	42.2	93.0	335	31.9
35-7	**93.0**	213	21.5	92.0	219	21.7
35-12	**90.5**	103	14.7	82.3	115	24.5
53-5	**97.5**	449	16.3	92.1	485	45.2
53-7	**97.8**	294	10.2	93.4	304	26
53-10	**94.5**	224	15.2	91.4	234	24.3
53-14	**91.2**	143	16.6	82.2	161	32
70-7	**95.9**	430	21.3	95.0	447	29.2
70-10	**95.4**	262	15.4	93.8	272	21
70-14	**93.2**	199	17.8	87.6	211	29.6
70-19	**90.7**	141	17.7	87.5	144	23.5
89-8	**80.6**	483	45.3	82.3	486	35.6
89-12	**96.2**	317	16.4	94.3	317	22.6
89-16	**98.8**	223	22.3	90.8	247	41.1
89-21	**90.6**	172	19.05	88.2	174	23.1
111-9	**97.2**	521	18.73	97.8	523	16.9
111-13	**96.1**	321	16.0	95.6	321	19.9
111-17	**94.0**	243	21.1	93.2	247	19.7
111-22	**91.7**	183	18.0	91.7	183	18
148-10	**98.0**	627	16.9	96.2	641	32.2
148-14	96.3	419	18.2	**96.4**	420	18.2
148-21	**95.2**	272	15.5	94.5	273	15.5
148-29	**90.8**	188	20.5	92.8	189	18
297-19	**97.3**	593	19.4	97.1	594	20.7
297-29	**94.2**	397	27.1	93.3	399	31.3
297-38	**94.2**	295	21.8	91.2	305	31
297-50	92.0	224	21.7	**92.4**	225	19.8

It is observed that PSO reports the solution at a faster rate than DE for all the datasets. However ,the algorithm structured based on DE provides reasonable good quality assignment of tasks and robots to workstations for large size problems in practical computational time but PSO algorithm could obtain better solution for all sets of problems at a faster rate. The computational time for DE could be on the higher side when compared to PSO due to repeated fitness value evaluation in case of selection

operation. In conclusion, PSO performs better in terms of quality of solution and computational time when compared to DE.

Table 8.3 Average Computational time of PSO and DE for straight RALB

Problems	Average Computational Time	
	PSO	DE
25	**5**	7
35	**17**	21
53	**21**	26
70	**55**	60
89	**61**	66
111	**185**	236
148	**355**	477
297	**990**	1007

8.2.5 Performance analysis for U-shaped RALB

This section presents the results obtained for U-shaped robotic assembly line for the objective of maximizing line efficiency by minimizing the workstation times. Both PSO and DE are used to find the solution for this type of RALB problem. Parameters used in the previous section are used for solving this problem. Thirty two problems available from the literature are used to test the performance of the proposed algorithms.

The results obtained by evaluating 32 test problems are presented in Table 8.4. Results reported are the best solution obtained for both the algorithms. Column I presents the problem evaluated, Column II, III and IV presents the line efficiency, cycle time and smoothness index of the best solution evaluated using PSO algorithm. Column V, VI and VII presents the results obtained using DE algorithm. Results reported are the best solution obtained for both the algorithms.

From the experiments conducted it is observed that 23 out of 32 datasets obtained same or better line efficiency for PSO when compared to DE. However, five datasets reported the same results for both the algorithms. Average improvement of only 1.05% in the line efficiency is obtained for PSO when compared to DE for small size datasets (up to 70 task problems) and the average improvement of line efficiency for large size datasets (above 89 task problems) when using PSO is found to be 0. 2%. Percentage could be low for both datasets because both PSO and DE reach solutions which are nearer to each other and few results are same for both the datasets. When comparing

the cycle time results obtained using PSO are better than DE for 21 datasets and 10 datasets reported the same cycle time. When comparing the smoothness index between two algorithms, it is observed that PSO produces better results for 21 datasets and 7 datasets produced same smoothness index. Smoothness index would vary even if the cycle time is same due to the variation in the workstation times.

Table 8.4 Line Efficiency of U-shaped RALB for PSO and DE

Work Station/ Robots	PSO			DE		
	Line Efficiency (%)	Cycle Time	Smoothness Index	Line Efficiency (%)	Cycle Time	Smoothness Index
25-3	**99.1**	500	6.02	**99.1**	500	6.02
25-4	**98**	294	4.1	91.5	318	33.5
25-6	**96.9**	183	7.3	**96.9**	183	7.3
25-9	**89.0**	109	15.9	88.9	109	15.8
35-4	**98.6**	345	3.6	**98.6**	345	3.6
35-5	96.7	333	11.8	**97.4**	334	11.8
35-7	94.6	210	14.2	**94.8**	215	13.5
35-12	**90.9**	103	13.8	87.3	106	17.8
53-5	**98.6**	443	8.02	97.3	459	16.7
53-7	**95.7**	283	16.9	93.8	286	21.8
53-10	**94.5**	215	16.3	93.7	220	16.4
53-14	**92.3**	141	15.4	90.0	148	18.3
70-7	**97.7**	427	11.8	97.4	427	12.2
70-10	**95.6**	262	15.1	94.0	266	22
70-14	91.7	197	23.9	**92.8**	199	18
70-19	88.9	140	19.9	**89.2**	140	17.9
89-8	83.4	476	19.9	**84.0**	475	19.9
89-12	94.5	312	19.7	**96.0**	315	16.4
89-16	**98.2**	222	21.5	**98.2**	224	22.03
89-21	**90.4**	169	24.2	88.6	172	25
111-9	96.7	519	19.6	**96.8**	520	23.6
111-13	**95.4**	317	19.2	94.4	319	23.5
111-17	**93.8**	242	18.3	**93.8**	242	18.3
111-22	90.0	181	21.4	**90.1**	181	22.1
148-10	**97.2**	619	19.9	96.7	629	26.3
148-14	**94.2**	416	27.3	94.0	418	32.2
148-21	92.6	269	21.2	**92.7**	275	22.5
148-29	**90.2**	187	22	89.9	187	22.5
297-19	**96.7**	588	24.5	96.4	589	24.5
297-29	**94.0**	389	32.7	93.4	390	34.2
297-38	**94.1**	288	20.3	93.6	291	22.9
297-50	**91.0**	222	21.1	90.9	222	22.7

Table 8.5 reports the average computational time for both the algorithms when evaluated for U-shaped robotic assembly line. It is observed that PSO performs at a faster rate than that of DE for almost all datasets. However, DE reports reasonably good quality solution for all the problems. Computational time for U-shaped RALB problems is higher due to the large search space.

The robustness and computational efficiency can be still improved if parameters are fine tuned. In conclusion, PSO performs better in terms of quality of solution and computational time when compared to DE.

Table 8.5 Average Computational time of PSO and DE for U-shaped RALB

Problems	Average Computational Time	
	PSO	DE
25	**7**	9
35	**18**	20
53	**35**	41
70	**59**	65
89	**103**	120
111	**245**	273
148	**415**	511
297	**1055**	1188

8.2.6 Comparison for straight and U-shaped RALB

The line efficiency and cycle time obtained for straight and U-shaped robotic assembly line using PSO and DE are compared when time data are used. Table 8.6 is formed by extracting the results obtained for the objective of maximizing the line efficiency for straight (Table 8.2) and U-shaped robotic assembly line (Table 8.4) using PSO and DE and Table 8.7 reports the cycle time obtained for both the layouts (straight and U-shaped) using PSO and DE.

U-shaped robotic assembly line produces better line efficiency for all the small size datasets (up to 70 task problems) when compared with straight robotic assembly line for both PSO and DE. For large size datasets, line efficiency reported using PSO is better for straight robotic assembly line when compared to the U-shaped assembly for most of the datasets in this group. For DE, eight problems in the group reports better efficiency for straight robotic assembly line when compared with U-shaped RALB. The variation in workstation time is lower when assembled through straight line, hence the line efficiency is higher for the large size dataset problems. However, from the tables it is observed that even though the efficiency for U-shaped is lower for these large size datasets, the cycle time for U-shaped RALB are still lower when compared to straight robotic assembly line.

In conclusion, U-shaped robotic assembly line performs better in case of line efficiency and cycle time for small size datasets. And for large size datasets, line efficiency is higher for straight robotic assembly line and cycle time is better for U-shaped robotic assembly line when the objective is to maximize the line efficiency by minimizing the workstation times.

Table 8.6 Comparison of Line Efficiency obtained using time data between straight and U-shaped RALB

Problem No:	Problem Dataset	Line Efficiency				Problem No:	Problem Dataset	Line Efficiency			
		PSO		DE				PSO		DE	
		Straight RALB	U-shaped RALB	Straight RALB	U-shaped RALB			Straight RALB	U-shaped RALB	Straight RALB	U-shaped RALB
1	25-3	97.3	**99.1**	97.3	**99.1**	17	89-8	80.6	**83.4**	82.3	**84**
2	25-4	97.1	**98**	88.6	**91.5**	18	89-12	**96.2**	94.5	94.3	**96**
3	25-6	90.5	**96.9**	88.2	**96.9**	19	89-16	**98.8**	98.2	90.8	**98.2**
4	25-9	87.4	**89**	84.5	**88.9**	20	89-21	**90.6**	90.4	88.2	**88.6**
5	35-4	99.4	98.6	98	**98.6**	21	111-9	**97.2**	96.7	**97.8**	96.8
6	35-5	95.2	**96.7**	93	**97.4**	22	111-13	**96.1**	95.4	**95.6**	94.4
7	35-7	93	**94.6**	92	**94.8**	23	111-17	**94**	93.8	93.2	**93.8**
8	35-12	90.5	**90.9**	82.3	**87.3**	24	111-22	**91.7**	90	**91.7**	90.1
9	53-5	97.5	**98.6**	92.1	**97.3**	25	148-10	**98**	97.2	96.2	**96.7**
10	53-7	97.8	95.7	93.4	**93.8**	26	148-14	**96.3**	94.2	**96.4**	94
11	53-10	94.5	**94.5**	91.4	**93.7**	27	148-21	**95.2**	92.6	**94.5**	92.7
12	53-14	91.2	**92.3**	82.2	**90**	28	148-29	**90.8**	90.2	**92.8**	89.9
13	70-7	95.9	**97.7**	95	**97.4**	29	297-19	**97.3**	96.7	**97.1**	96.4
14	70-10	95.4	**95.6**	93.8	**94**	30	297-29	**94.2**	94	93.3	93.4
15	70-14	**93.2**	91.7	87.6	**92.8**	31	297-38	**94.2**	94.1	91.2	93.6
16	70-19	**90.7**	88.9	87.5	**89.2**	32	297-50	**92**	91	**92.4**	90.9

Table 8.7 Comparison of Cycle time obtained using time data between straight and U-shaped RALB

Problem No:	Problem Dataset	Cycle Time				Problem No:	Problem Dataset	Cycle Time			
		PSO		DE				PSO		DE	
		Straight RALB	U-shaped RALB	Straight RALB	U-shaped RALB			Straight RALB	U-shaped RALB	Straight RALB	U-shaped RALB
1	25-3	503	**500**	503	**500**	17	89-8	483	**476**	486	**475**
2	25-4	294	**294**	329	**318**	18	89-12	317	**312**	317	**315**
3	25-6	200	**183**	208	**183**	19	89-16	223	**222**	247	**224**
4	25-9	110	**109**	114	**109**	20	89-21	172	**169**	174	**172**
5	35-4	342	**345**	347	**345**	21	111-9	521	**519**	523	**520**
6	35-5	329	**333**	335	**334**	22	111-13	321	**317**	321	**319**
7	35-7	213	**210**	219	**215**	23	111-17	243	**242**	247	**242**
8	35-12	103	**103**	115	**106**	24	111-22	183	**181**	183	**181**
9	53-5	449	**443**	485	**459**	25	148-10	627	**619**	641	**629**
10	53-7	294	**283**	304	**286**	26	148-14	419	**416**	420	**418**
11	53-10	224	**215**	234	**220**	27	148-21	272	**269**	273	**275**
12	53-14	143	**141**	161	**148**	28	148-29	188	**187**	189	**187**
13	70-7	430	**427**	447	**427**	29	297-19	593	**588**	594	**589**
14	70-10	262	**262**	272	**266**	30	297-29	397	**389**	399	**390**
15	70-14	199	**197**	211	**199**	31	297-38	295	**288**	305	**291**
16	70-19	141	**140**	144	**140**	32	297-50	224	**222**	225	**222**

8.3 Line Efficiency calculation using energy data

Objective of this problem is to maximize the line efficiency by minimizing the energy consumption in a robotic assembly line. Till date no research has been reported on the objective of optimizing the line efficiency when the energy consumption is minimized. The objective function evaluated for this research for straight and U-shaped type of assembly line is Line Efficiency (LE). The section below explains how the line efficiency is calculated in a straight and U-shaped robotic assembly line when energy consumption at workstations is minimized. The steps involved in the line efficiency calculation are explained below.

Steps involved in Line Efficiency Calculation in straight RAL

i. Tasks are allocated to the workstation based on the energy consumption data (Section 6.1.1) in a straight robotic assembly line. The procedure tries to allocate the tasks to the workstation by minimizing the energy consumption at each workstation.

ii. The best available robot which performs the allotted tasks with minimum energy consumption is allotted to the workstation.

iii. Based on the tasks and robot allocated to the workstation, workstation times are calculated using the robot performance time data.

iv. The workstation with the maximum workstation time is the cycle time of the allocation.

v. Line efficiency is calculated using Equation 8.1. Line efficiency is the direct indication of the efficiency of a given assembly line.

Figure 8.3 shows the tasks and robot allocation for the given sequence of tasks. Energy consumed and workstation times calculated are presented in the figure. Allocation is done based on the objective of minimizing the energy consumption allocation and workstation times calculated on procedure for a straight robotic assembly line. The workstation times are used to calculate the line efficiency.

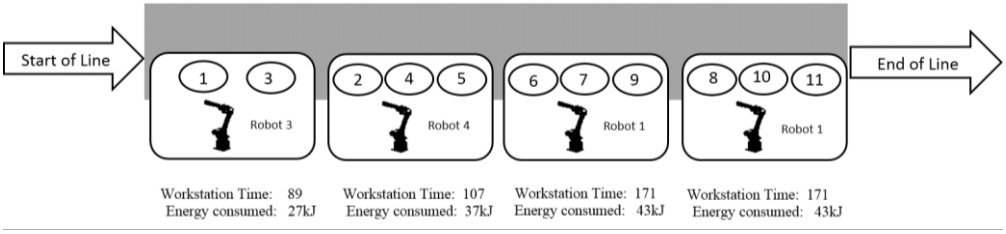

Figure 8.3 Workstation times and energy consumption in straight RALB

Straight line robotic assembly line's efficiency is calculated as follows:

$$LE = \frac{\sum_{k=1}^{N_w} S_k}{N_w * c} * 100 = (89+107+171+171)/\ (4*171)*100=78.65\%$$

Similarly, for U-shaped robotic assembly line procedure to evaluate the line efficiency is same except for the task allocation. The tasks are allocated from either side of the sequence. Section 6.2.2 explains the procedure of tasks and robot allocation in a U-shaped robotic assembly line. The tasks and robots are allocated with an objective of minimizing the energy consumption at each workstation. Figure 8.4 shows the allocation done based on energy consumption data. The details of the workstation times and energy consumed are presented in the graph. Using the workstation times, the line efficiency is calculated.

Line Efficiency: (75+113+115+156) / (4*156)*100 = 74%.

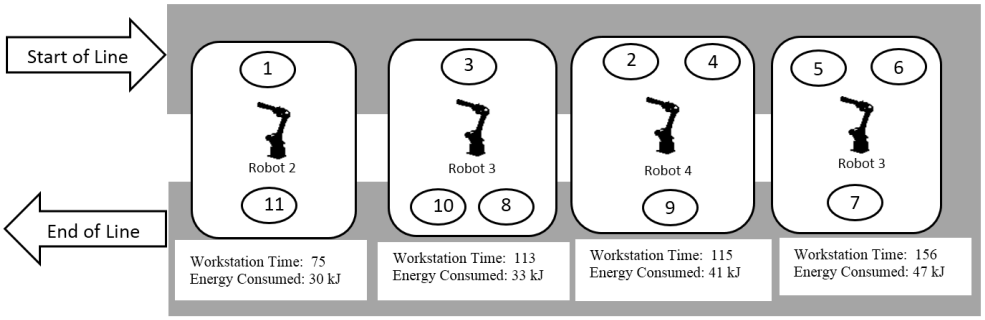

Figure 8.4 Workstation times and energy consumption in U-shaped RALB

8.3.1 Performance in straight RALB for energy data

Straight robotic assembly line is evaluated using PSO and DE with the objective of maximizing the line efficiency when the energy consumption is minimized. Energy data is used for the allocation of tasks and robots, using the allocation details the workstation time and cycle time of the assembly line are calculated. The workstation time and cycle time is used for calculating the line efficiency of the assembly line. All the 32 test problems are evaluated using the proposed PSO and DE algorithm. The non-deterministic nature of the algorithm and problem makes it necessary to run same problem multiple times. All the problems are run for ten times and it is observed that most of the runs converged to same solution. The parameters used for evaluating the line efficiency in Section 8.3.3 is used for evaluating the line efficiency in this section also. The results reported are the best solutions obtained for both the algorithms for the objective of maximizing the line efficiency. Best results obtained by evaluating 32 test problems using PSO and DE are presented in Table 8.8. Column I presents the problem evaluated, Column II, III and IV presents the line efficiency, energy consumption and cycle time of the best solution evaluated using PSO algorithm. Column V, VI and VII presents the results obtained using DE algorithm.

From the experiments conducted it is observed that 30 out of 32 datasets obtained same or better line efficiency for DE when compared to PSO. And one dataset reported the same results for both the algorithms. When comparing the results between PSO and DE, it is observed that DE produces results with an average improvement in the line efficiency of 7.8 % for small size datasets (up to 70 task problems) and the average improvement in the line efficiency for large size datasets is found to be 8.1%.

Table 8.8 Line Efficiency of straight RALB for PSO and DE using energy data

Work Station/ Robots	PSO			DE		
	Line Efficiency (%)	Energy Consumption (kJ)	Cycle Time	Line Efficiency (%)	Energy Consumption (kJ)	Cycle Time
25-3	98.1	489	521	**98.4**	502	536
25-4	85.3	353	342	**90.3**	343	329
25-6	88.0	357	207	**88.0**	357	207
25-9	74.5	235	143	**76.4**	242	147
35-4	**90.2**	1037	518	80.9	1045	516
35-5	76.7	900	444	**96.4**	890	357
35-7	77.5	1000	343	**85.8**	989	314
35-12	**68.7**	688	153	73.9	687	130
53-5	**74.1**	2680	605	75.7	2680	590
53-7	69.9	1970	439	**90.6**	1970	345
53-10	80.7	2186	308	**83.6**	2148	263
53-14	73.6	2016	202	**79.8**	2050	183
70-7	94.2	4093	620	**94.4**	4191	633
70-10	88.5	3046	290	**90.3**	3106	302
70-14	67	3815	338	**83.2**	3766	256
70-19	63.1	3243	232	**71.2**	3225	204
89-8	75.4	4956	562	**77.0**	4922	562
89-12	80.0	5509	438	**81.3**	5499	438
89-16	83.2	4906	288	**92.6**	4887	265
89-21	72.5	4150	236	**74.3**	4182	232
111-9	84.11	7149	529	**92.2**	7131	674
111-13	82.1	7030	396	**89.0**	7137	391
111-17	88.3	6857	280	**92.1**	6877	266
111-22	72.3	6630	255	**87.7**	6667	215
148-10	94.6	9798	688	**95.8**	9798	678
148-14	88.7	10524	461	**90.7**	10645	466
148-21	75.3	10084	384	**83.5**	9927	335
148-29	57.1	8334	317	**68.3**	8508	263
297-19	78.3	24518	809	**88.7**	24351	688
297-29	**87.4**	24554	458	83.8	24404	528
297-38	84.2	22485	409	**86.1**	22482	348
297-50	64.8	21089	348	**80.4**	21050	336

The results reported for the line efficiency is calculated by minimizing the energy consumption of the workstation. When comparing the energy consumption for the two algorithms, it is observed that the energy consumption is lower for 21 datasets and for the other datasets PSO reported lower energy consumption. 25 datasets reported same or better cycle time for DE when compared with PSO. Even though cycle time is low for 7 datasets for PSO, line efficiency is still better for DE. This is due to the variation of workstation times which affects the line efficiency.

Table 8.9 reports the average computational time for both the algorithms when evaluated for straight robotic assembly line using energy data. DE performs at a faster

rate when compared to PSO for almost all datasets. However, PSO also reports results at a reasonable computational time. Computational time for large problem (297 tasks) is high for DE. This could be due to the large size of the problem and repeated selection process of DE. Computational performance of PSO can be improved if parameters are further fine-tuned. In conclusion, DE performs better than PSO in terms of quality of the solution and computational time.

Table 8.9 Average Computational time of PSO and DE for straight RALB using energy data

Problems	Average Computational Time	
	PSO	DE
25	9	**8**
35	23	**20**
53	41	**36**
70	73	**67**
89	95	**92**
111	288	**281**
148	550	**540**
297	**1700**	1739

8.3.2 Performance analysis in U-shaped RALB for energy data

Best results obtained for PSO and DE for the objective of maximizing the line efficiency when the energy consumption is minimized is presented in Table 8.10. The parameters used for evaluating the line efficiency in Section 8.3.3 is used for evaluating the line efficiency in this section also. Similar to Table 8.8, Column I presents the problem evaluated, Column II, III and IV presents the line efficiency, energy consumption and cycle time of the best solution evaluated using PSO algorithm. Column V, VI and VII presents the results obtained using DE algorithm in Table 8.10.

It is observed from the experiments conducted, 28 out of 32 datasets better line efficiency for DE when compared to PSO. DE obtained average improvement of 6.2 % in the line efficiency when compared to PSO for small size datasets (up to 70 task problems) and the improvement of line efficiency for large size datasets when using DE is found to be 2.0%.

Table 8.10 Line Efficiency of U-shaped RALB for PSO and DE using energy data

Work Station/ Robots	PSO			DE		
	Line Efficiency (%)	Energy Consumption (kJ)	Cycle Time	Line Efficiency (%)	Energy Consumption (kJ)	Cycle Time
25-3	85.6	488	561	**98.6**	493	496
25-4	87.7	336	334	**90.7**	341	320
25-6	85.6	358	230	**87.3**	363	230
25-9	**72.7**	242	148	72.6	244	154
35-4	79.2	1010	535	**82.4**	1045	477
35-5	87.1	855	397	**94.4**	909	380
35-7	87.2	983	307	**87.5**	998	302
35-12	81.2	653	125	**86.4**	681	122
53-5	77.08	2638	610	**96.3**	2680	464
53-7	89.7	1959	330	**95.8**	1970	326
53-10	83.6	2133	260	**91.3**	2150	249
53-14	80.2	1990	183	**83.6**	2003	183
70-7	94.6	4256	640	**95.7**	4364	656
70-10	87.7	3013	298	**89.3**	3050	294
70-14	72.4	3700	272	**78.9**	3739	266
70-19	79.0	3063	170	**81.0**	3223	176
89-8	77.1	4926	558	**77.3**	4991	545
89-12	80.2	5496	426	**82.7**	5537	424
89-16	81.4	4753	285	**90.7**	4798	269
89-21	82.7	4135	200	**84.9**	4158	201
111-9	**95.1**	7138	696	90.4	7172	688
111-13	83.3	6818	378	**89.3**	7078	377
111-17	89.9	6688	266	**92.5**	6804	265
111-22	**83.9**	6518	217	80.8	6538	223
148-10	94.8	9798	671	**96.4**	9798	664
148-14	**90.2**	10395	458	85.7	10732	521
148-21	83.8	10235	394	**84.8**	9894	318
148-29	86.5	8186	200	**88.3**	8296	198
297-19	88.3	24313	696	**88.8**	24320	695
297-29	67.0	24302	592	**78.9**	24456	500
297-38	85.3	22037	350	**85.9**	22585	357
297-50	86.1	20647	255	**86.4**	20888	256

Percentage could be low for large size datasets because both PSO and DE could reach solutions which are nearer to each other. The results reported for the line efficiency is calculated by minimizing the energy consumption of the workstation. From the table, cycle time obtained for both DE and PSO are reported along with the energy consumption of the assembly line. It could be seen that cycle time is lower in case of DE solution for 25 out of 32 problems. When comparing the energy consumption amongst the two models, it could be seen that PSO is getting lower energy consumption compared to that of DE for 31 out of 32 problems addressed here.

Table 8.11 reports the computational time for both the algorithms. The algorithm structured based on PSO provides reasonable quality assignment of tasks and robots to workstations for small size problems in practical computational time but DE algorithm could obtain the solution for the same set of solution at a faster rate. In case of large size datasets (148 and 297 task problems), DE algorithm takes more computational time than PSO. The computational time for DE could be on the higher side when compared to PSO due to repeated fitness value evaluation in case of selection operation. Results show that DE is better for larger size problems in terms of line efficiency, its robustness and computational efficiency can be improved by fine tuning the parameters. In conclusion, DE performs better than PSO in terms of quality of the solution and computational time.

Table 8.11 Average Computational time of PSO and DE for U-shaped RALB using energy data

Problems	Average Computational Time	
	PSO	DE
25	11	**9**
35	26	**22**
53	44	**38**
70	75	**70**
89	99	**97**
111	**295**	317
148	**560**	585
297	**1728**	1820

8.3.3 Comparison of straight and U-shaped RALB for energy data results

The line efficiency and cycle time obtained for straight and U-shaped robotic assembly line using PSO and DE are compared when energy data are used. Table 8.12 is formed by extracting the results obtained for the objective of maximizing the line efficiency for straight and U-shaped robotic assembly line using PSO and DE, Table 8.13 reports the cycle time obtained for both the layouts using PSO and DE and Table 8.14 reports the energy consumption calculated for both the layouts.

Eleven problems in the small size datasets groups reported better efficiency for U-shaped robotic assembly line when compared with straight robotic assembly line for DE. And in large size dataset group, ten problems reported better efficiency for U-shaped robotic assembly line.

Table 8.12 Comparison of Line Efficiency obtained using energy data between straight and U-shaped RALB

Problem No:	Problem Dataset	Line Efficiency				Problem No:	Problem Dataset	Line Efficiency			
		PSO		DE				PSO		DE	
		Straight RALB	U-shaped RALB	Straight RALB	U-shaped RALB			Straight RALB	U-shaped RALB	Straight RALB	U-shaped RALB
1	25-3	**98.1**	85.6	98.4	**98.6**	17	89-8	75.4	**77.1**	77	**77.3**
2	25-4	85.3	**87.7**	90.3	**90.7**	18	89-12	80	**80.2**	81.3	**82.7**
3	25-6	**88**	85.6	**88.0**	87.3	19	89-16	**83.2**	81.4	92.6	**90.7**
4	25-9	**74.5**	72.7	**76.4**	72.6	20	89-21	72.5	**82.7**	74.3	**84.9**
5	35-4	**90.2**	79.2	80.9	**82.4**	21	111-9	84.11	**95.1**	**92.2**	90.4
6	35-5	76.7	**87.1**	96.4	94.4	22	111-13	82.1	**83.3**	89	**89.3**
7	35-7	77.5	**87.2**	85.8	**87.5**	23	111-17	88.3	**89.9**	92.1	**92.5**
8	35-12	68.7	**81.2**	73.9	**86.4**	24	111-22	72.3	**83.9**	**87.7**	80.8
9	53-5	74.1	**77.0**	75.7	**96.3**	25	148-10	94.6	**94.8**	95.8	**96.4**
10	53-7	69.9	**89.7**	90.6	**95.8**	26	148-14	88.7	**90.2**	**90.7**	85.7
11	53-10	80.7	**83.6**	83.6	**91.3**	27	148-21	75.3	**83.8**	83.5	**84.8**
12	53-14	73.6	**80.2**	79.8	**83.6**	28	148-29	57.1	**86.5**	68.3	**88.3**
13	70-7	94.2	**94.6**	94.4	**95.7**	29	297-19	78.3	**88.3**	88.7	**88.8**
14	70-10	**88.5**	87.7	**90.3**	89.3	30	297-29	**87.4**	67	**83.8**	78.9
15	70-14	67	**72.4**	**83.2**	78.9	31	297-38	84.2	**85.3**	**86.1**	85.9
16	70-19	63.1	**79.0**	71.2	**81.0**	32	297-50	64.8	**86.1**	80.4	**86.4**

When comparing the line efficiency between the straight and U-shaped RALB for PSO, it is observed that line efficiency of U-shaped is better for eleven datasets in the small dataset group and fourteen datasets in the large dataset group reports better line efficiency for U-shaped. So, it is clearly seen that U-shaped RALB performs better in terms of line efficiency when the allocation is done based on the energy based data where the objective is to minimize the energy consumption of the assembly line.

Table 8.13 provides a comparison for cycle time obtained for both the algorithms. For small size datasets when the results are obtained using PSO, U-shaped RALB obtains lower results for nine problems in the group when compared with straight robotic assembly line and when cycle time is compared for same datasets for DE, it is observed that eleven problems obtains lower cycle time than straight robotic assembly line. For large size datasets using PSO, U-shaped RAL reports better solution for thirteen datasets when compared with straight RAL. And when the cycle time is compared for DE for the large size datasets, ten datasets in the group reported lower cycle time for U-shaped RALB.

Table 8.13 Comparison of cycle time obtained using energy data between straight and U-shaped RALB

Problem No:	Problem Dataset	Cycle Time				Problem No:	Problem Dataset	Cycle Time			
		PSO		DE				PSO		DE	
		Straight RALB	U-shaped RALB	Straight RALB	U-shaped RALB			Straight RALB	U-shaped RALB	Straight RALB	U-shaped RALB
1	25-3	**521**	561	536	**496**	17	89-8	562	**558**	562	**545**
2	25-4	342	**334**	329	**320**	18	89-12	438	**426**	438	**424**
3	25-6	**207**	230	**207**	230	19	89-16	288	**285**	**265**	269
4	25-9	**143**	148	**147**	154	20	89-21	236	**200**	232	**201**
5	35-4	**518**	535	516	**477**	21	111-9	**529**	696	**674**	688
6	35-5	444	**397**	**357**	380	22	111-13	396	**378**	391	**377**
7	35-7	343	**307**	314	**302**	23	111-17	280	**266**	266	**265**
8	35-12	153	**125**	130	**122**	24	111-22	255	**217**	**215**	223
9	53-5	**605**	610	590	**464**	25	148-10	688	**671**	678	**664**
10	53-7	439	**330**	345	**326**	26	148-14	461	**458**	**466**	521
11	53-10	308	**260**	263	**249**	27	148-21	**384**	394	335	**318**
12	53-14	202	**183**	183	**183**	28	148-29	317	**200**	263	**198**
13	70-7	**620**	640	**633**	656	29	297-19	809	**696**	**688**	695
14	70-10	**290**	298	302	**294**	30	297-29	**458**	592	528	**500**
15	70-14	338	**272**	**256**	266	31	297-38	**409**	350	**348**	357
16	70-19	232	**170**	204	**176**	32	297-50	348	**255**	336	**256**

Energy consumption reported here are obtained by taking the sum of energy consumption at each workstation. From Table 8.14, for small size datasets U-shaped RALB reports lesser energy consumption for thirteen datasets when the results are taken using PSO and when the results are taken based on DE, energy consumption is lower in U-shaped RALB for eight datasets. Even though only eight datasets are lower for DE for small size datasets, the line efficiency is better in these problems for U-shaped. For large size datasets, results obtained using PSO reports that fifteen datasets reports lower energy consumption for U-shaped and when the energy consumption is compared for DE, it is observed that U-shaped reports lower energy consumption for eleven datasets when compared with the energy consumption of straight RALB.

In conclusion, U-shaped robotic assembly line performs better in terms of line efficiency, cycle time and energy consumption for most of the problems in both small and large size datasets when compared with straight robotic assembly line when the

objective of maximizing the line efficiency is done by minimizing the energy consumption.

Table 8.14 Comparison of energy consumption obtained between straight and U-shaped RALB using energy data

Problem No:	Problem Dataset	Energy Consumption(kJ)				Problem No:	Problem Dataset	Energy Consumption(kJ)			
		PSO		DE				PSO		DE	
		Straight RALB	U-shaped RALB	Straight RALB	U-shaped RALB			Straight RALB	U-shaped RALB	Straight RALB	U-shaped RALB
1	25-3	489	**488**	502	**493**	17	89-8	4956	**4926**	4922	**4991**
2	25-4	353	**336**	343	**341**	18	89-12	5509	**5496**	5499	5537
3	25-6	357	358	**357**	363	19	89-16	4906	**4753**	4887	**4798**
4	25-9	**235**	242	**242**	244	20	89-21	4150	**4135**	4182	**4158**
5	35-4	1037	**1010**	**1045**	**1045**	21	111-9	7149	**7138**	7131	7172
6	35-5	900	**855**	**890**	909	22	111-13	7030	**6818**	7137	**7078**
7	35-7	1000	**983**	**989**	998	23	111-17	6857	**6688**	6877	**6804**
8	35-12	688	**653**	687	**681**	24	111-22	6630	**6518**	6667	**6538**
9	53-5	2680	**2638**	**2680**	**2680**	25	148-10	**9798**	**9798**	**9798**	**9798**
10	53-7	1970	**1959**	**1970**	**1970**	26	148-14	10524	**10395**	**10645**	10732
11	53-10	2186	**2133**	2148	2150	27	148-21	**10084**	10235	9927	**9894**
12	53-14	2016	**1990**	2050	**2003**	28	148-29	8334	**8186**	8508	**8296**
13	70-7	4093	4256	**4191**	4364	29	297-19	24518	**24313**	24351	**24320**
14	70-10	3046	**3013**	3106	**3050**	30	297-29	24554	**24302**	**24404**	24456
15	70-14	3815	**3700**	3766	**3739**	31	297-38	22485	**22037**	**22482**	22585
16	70-19	3243	**3063**	3225	**3223**	32	297-50	21089	**20647**	21050	**20888**

8.4 Summary

Line efficiency is one of the important objectives to optimize in assembly lines. Line efficiency helps to analyze if all the resources available are utilized efficiently. Industries can produce more products in shorter time when the line efficiency is high. Two models to calculate the efficiency are presented in this chapter. The first model is developed based on the objective of minimizing the workstation times for a robotic assembly line using the time data. The two layouts (straight and U-shaped) of robotic assembly line are tested for the performance. The line efficiency obtained for the two layouts are compared along with the cycle time and smoothness index of the best solution obtained. Particle swarm optimization (PSO) and Differential evolution (DE) are the two metaheuristic algorithms used as the optimization tool to solve this problem for both the layouts (straight and U-shaped). Among the two optimization tools it is observed that PSO performs better for both the layouts. From the results it is concluded that U-shaped robotic assembly line performs

better in case of line efficiency and cycle time for small size datasets. And for large size datasets, line efficiency is higher for straight robotic assembly line and cycle time is better for U-shaped robotic assembly line when the objective is to maximize the line efficiency by minimizing the workstation times.

Second model developed is based on the objective of minimizing the energy consumption for a robotic assembly line. The line efficiency of the assembly line which performs the tasks with minimum energy consumption is evaluated. The line efficiency is calculated and compared for both the layouts (straight and U-shaped). Particle swarm optimization (PSO) and Differential evolution (DE) are the two metaheuristic algorithms used as the optimization tool to solve this problem for both the layouts. It is observed that DE performs better for both the layouts for most of the datasets. The comparison of results obtained for both the layouts are presented. It is observed that DE performs better in terms of the objective function when compared to PSO. It is concluded that U-shaped robotic assembly line performs better in terms of line efficiency, cycle time and energy consumption for most of the problems in both small and large size datasets.

CHAPTER 9

Conclusion

A summary of research work conducted on robotic assembly line balancing problems to optimize different objectives are presented in this thesis. The major contributions made and the scope of future work is also presented.

The literature survey revealed that researches on assembly line balancing problems have been conducted extensively and different optimization methods are developed in the past. Most of the methods are designed for balancing manual assembly lines. Detailed literature survey for different types of assembly line balancing problems and different optimization techniques are presented in Chapter 2. It is observed that, only few literatures could be found on robotic assembly line balancing problems. The research gaps are identified and the research objectives are framed. Different research objectives to be addressed in thesis are presented in Chapter 3. Mathematical models for different RALB problems are presented along with the assumptions considered to solve these problems.

9.1 RALB problem to minimize cycle time

RALB problem is developed and solved to optimize the cycle time of a robotic assembly line. The procedures and results obtained for this model are presented in Chapter 5. The existing literature survey is comprehensively analysed to identify if available solution could be improved. IBM Cplex Optimization studio Version 12.6.0.0 standard optimization software is used to solve the problems to get optimal solutions. It is observed that only fourteen problems in the datasets could be solved within an acceptable time span. Two allocation procedures are implemented to improve the quality of the available solution. PSO is proposed to solve the problem. Results of the proposed models are improved using a local exchange procedure. Out of the two allocation procedures, consecutive allocation procedure performs better. PSO variants and hybrid PSO algorithms are also proposed to solve the problem. Variants are developed based on the variation in the velocity update in PSO and hybrid models are developed by hybridizing with GA (breeding) and Cuckoo Search. The performances of the variants and hybrid algorithms of PSO are tested on benchmark problems and the

obtained results are compared with the reported results. Set of experiments are conducted to investigate the effects of the parameters on the solution quality. Results show that the proposed hybrid CS-PSO algorithm reports better solution than the solution reported in the literature. The robotic assembly line considered is a straight assembly line problem where the tasks and workstations are arranged in a straight line.

A new robotic U-shaped assembly line balancing (RUALB) problem is also presented. No work has been reported on U-shaped robotic assembly line. The major objective of the problem developed is to minimize the cycle time of the assembly line when the assembly line can be arranged for a U-shaped configuration. Allocation of tasks and robots in U-shaped configuration is highly complex when compared to the straight assembly line. Tasks are assigned to the workstations when either all of their predecessors or all of their successors have already been assigned to workstations. PSO algorithm is proposed to solve this problem. Thirty-two benchmark problems originally proposed by earlier researchers to solve RALB were adopted to the test the performance of RUALB. Extensive computational experiments were conducted and the results are reported in Chapter 5. From the results, it is observed that the cycle time of U-shaped robotic assembly line is lower than the straight robotic assembly line. To measure the complexity of the problem different complexity measures are used and the results are reported in this chapter.

9.2 RALB problem to minimize energy consumption

These days, manufacturing industries gives utmost importance for reducing the energy consumption due to the increasing energy cost and fast depletion of energy sources. The industries need to reduce the energy consumption to improve their profitability. Models are developed with an objective of minimizing energy consumption in a robotic assembly line. From the literature, it is observed that there has been no previous work reported on optimizing energy consumption in a robotic assembly line. Particle swarm optimization algorithm is proposed to solve the proposed models. Datasets to solve the proposed model is developed by embedding the energy data into the benchmark datasets. Using the developed energy data and existing time datasets, the energy consumption and the subsequent cycle time is evaluated for both the layouts. The energy consumption of the robotic assembly line when the allocation of tasks and robots are done based on the objective of minimizing the cycle time is also

calculated. This is proposed to check the energy saving capacity in the model proposed when the allocation of tasks and robots are performed. Several computational experiments are conducted for both the layouts and the best solution obtained for each models are presented in Chapter 6. A comparative analysis is done on the straight and U-shaped robotic assembly line to check which layout is better. From the study, it is concluded that U-shaped performs better for both the objectives of minimizing cycle time and energy consumption.

9.3 RALB problem to minimize assembly line cost

A new robotic assembly line balancing problem with an objective of minimizing assembly line cost is developed. Robot needs to be optimally assigned to the work stations such a way that the cost to perform these allocated tasks by robots be the minimum. Tasks and robots are allocated based on consecutive allocation heuristic. From the literature, it is observed that there has been no previous work reported on optimizing assembly line cost of a robotic assembly line. NP-hard nature of the problem makes it necessary to implement different metaheuristic algorithms (PSO Variants and DE) to solve the problem. Datasets are randomly generated by using the existing benchmark datasets. Results obtained by metaheuristic algorithms are tested on the datasets developed. It is observed that DE performs better than PSO in terms of the quality of the solution.

Sample problems are used to find the parameters to be used in the algorithms. Cycle time is also calculated from the cost and the performance is compared. It can be clearly concluded that U-shaped layout performs better than the straight robotic assembly line layout for the objective of minimizing the assembly line cost. Assembly line cost is also calculated for the problems when the allocation is conducted based on the objective of minimizing the cycle time. Both assembly line cost and cycle time for both the layouts are compared and it is concluded from the results obtained that U-shaped robotic assembly line reports better cycle time and assembly line cost when compared with straight robotic assembly line. Industrial managers can chose any of the models based on the priorities and demands.

9.4 RALB problem to maximize line efficiency

Two models for evaluating the line efficiency of the robotic assembly line are proposed. First model aims at maximizing the line efficiency by minimizing the workstation times and the second model aims at maximizing the line efficiency by minimizing the energy consumption of the workstations. The two models are solved for both layouts (straight and U-shaped) of robotic assembly line using PSO and DE algorithms. Extensive experiments are conducted on the benchmark data. Sample problems are used to fix the parameters to be used for the algorithms. Based on the comparative analysis, it is observed that for the first model, line efficiency is better for U-shaped robotic assembly line and line efficiency is better for straight line for the large size datasets. This is due to the variation of workstation times which affects the line efficiency. PSO and DE algorithm showed the same trend in the results. The cycle time and smoothness index are also evaluated and compared.

From the analysis on the second model, it is observed that for both PSO and DE, U-shaped robotic assembly line performs better in terms of line efficiency when compared to straight robotic assembly line. The cycle times of the problems tested along with the energy consumption for both the layouts are compared and it is concluded that U-shaped robotic assembly line performs better.

9.5 Contribution of this thesis

Efficient metaheuristic algorithms are proposed and solved for:

i. Robotic assembly line balancing problem with the objective of minimizing the cycle time for straight and U-shaped robotic assembly line.
ii. Energy based robotic assembly line problem for both straight and U-shaped robotic assembly line with an objective of minimizing the energy consumption.
iii. Cost based robotic assembly line balancing problem in both straight and U-shaped robotic assembly line with an objective of minimizing the total assembly line cost.
iv. Robotic assembly line problem with an objective of maximising the line efficiency by minimizing workstation times and minimizing energy consumption at each workstation.

9.6 Limitations of this research

The limitations of this research work are that only one robot can be assigned to one workstation. Assigned robot cannot handle multiple workstations. Tasks which are assigned to the workstation cannot be split. Computational time increases significantly when the problem size increases.

9.7 Future Research Proposals

Few areas of further research in RALB problems are:

i. Different other efficient metaheuristics can also be applied for the presently developed RALB problems.
ii. Robotic assembly line could be designed for two-sided and parallel assembly line.
iii. The models proposed in thesis are for a single model, robotic assembly lines could be designed for assembly of mixed and multi products.
iv. For energy based robotic assembly line balancing problems, algorithm could be tested on a specific time horizon where factors such as maintenance operation and effect of failures of the resources in the system could be included. The planning horizon can also be included in the model.

REFERENCES

Aase, G. R., Olson, J. R. & Schniederjans, M. J. 2004. U-shaped assembly line layouts and their impact on labor productivity: An experimental study. *European Journal of Operational Research,* 156**,** 698-711.

Aigbedo, H. & Monden, Y. 1997. A parametric procedure for multicriterion sequence scheduling for JustIn-Time mixed-model assembly lines. International Journal of Production Research, 35, 2543-2564.

Ajenblit, D. A. & Wainwright, R. L. 1998. Applying genetic algorithms to the U-shaped assembly line balancing problem. In: Proceedings of World Congress on Computational Intelligence, Anchorage, Alaska. IEEE, 96-101.

Ali, M. M. & Törn, A. 2004. Population set-based global optimization algorithms: some modifications and numerical studies. Computers & Operations Research, 31, 1703-1725.

Alp, A. 2004. Ant colony optimization for the single model U-type assembly line balancing problem. PhD Thesis, Bilkent University.

Amen, M. 2000a. An exact method for cost-oriented assembly line balancing. International Journal of Production Economics, 64, 187-195.

Amen, M. 2000b. Heuristic methods for cost-oriented assembly line balancing: A survey. International Journal of Production Economics, 68, 1-14.

Amen, M. 2001. Heuristic methods for cost-oriented assembly line balancing: A comparison on solution quality and computing time. International Journal of Production Economics, 69, 255-264.

Angeline, P. J. 1998. Evolutionary optimization versus particle swarm optimization: Philosophy and performance differences. In: Proceedings of Evolutionary Programming VII, Springer, 601-610.

Arcus, A. L. 1965. A computer method of sequencing operations for assembly lines. International Journal of Production Research, 4, 259-277.

Arumugam, M. S., Rao, Mvc , Aarthi,Chandramohan 2008. A new and improved version of particle swarm optimization algorithm with global–local best parameters. Knowledge and Information systems, 16, 331-357.

Avikal, S., Jain, R., Mishra, P. & Yadav, H. 2013. A heuristic approach for U-shaped assembly line balancing to improve labor productivity. Computers & Industrial Engineering, 64, 895-901.

Bartholdi, J. 1993. Balancing two-sided assembly lines: A case study. International Journal of Prodction Research, 31, 2447-2461.

Bautista, J. & Pereira, J. 2002. Ant algorithms for assembly line balancing. Ant algorithms. Springer.

Baybars, I. 1986. A survey of exact algorithms for the simple assembly line balancing problem. Management science, 32, 909-932.

Baykasoğlu, A. 2006. Multi-rule multi-objective simulated annealing algorithm for straight and U type assembly line balancing problems. Journal of Intelligent Manufacturing, 17, 217-232.

Baykasoğlu, A. & Özbakır, L. 2007. Stochastic U-line balancing using genetic algorithms. The International Journal of Advanced Manufacturing Technology, 32, 139-147.

Becker, C. & Scholl, A. 2006. A survey on problems and methods in generalized assembly line balancing. European Journal of Operational Research, 168, 694-715.

Bertoldi, P., Aebischer, B., Edlington, C., Hershberg, C., Lebot, B., Lin, J., Marker, T., Meier, A., Nakagami, H. & Shibata, Y. 2002. Standby power use: How big is the problem? What policies and technical solutions can address it? Lawrence Berkeley National Laboratory.

Bhattacharjee, T. & Sahu, S. 1990. Complexity of single model assembly line balancing problems. Engineering costs and production economics, 18, 203-214.

Blondin, J. 2009. Particle swarm optimization: A tutorial. from site: http://cs.armstrong.edu/saad/csci8100/pso_tutorial.pdf

Boeing. 2014. " Boeing 777 Moving Production Line". Boeing. Accessed October 29. http://www.boeing.com/boeing/commercial/777family/777movingline.page

Boussaïd, I., Lepagnot, J. & Siarry, P. 2013. A survey on optimization metaheuristics. Information Sciences, 237, 82-117.

Boysen, N., Fliedner, M. & Scholl, A. 2007. A classification of assembly line balancing problems. European Journal of Operational Research, 183, 674-693.

Bukchin, J. & Tzur, M. 2000. Design of flexible assembly line to minimize equipment cost. Iie Transactions, 32, 585-598.

Burnwal, S. & Deb, S. 2013. Scheduling optimization of flexible manufacturing system using cuckoo search-based approach. The International Journal of Advanced Manufacturing Technology, 64, 951-959.

Capacho Betancourt, L. 2008. ASALBP: the alternative subgraphs assembly line balancing problem. Formalization and resolution procedures. PhD Thesis, Polytechnic University of Catalonia. Institute of Industrial and Control Engineering.

Chiang, W.-C. 1998. The application of a tabu search metaheuristic to the assembly line balancing problem. Annals of Operations Research, 77, 209-227.

Chong, K. E., Omar, M. K. & Bakar, N. A. 2008. Solving assembly line balancing problem using genetic algorithm with heuristics-treated initial population. In: Proceedings of World Congress on Engineering, London, U.K.

Chryssolouris, G. 2005. Manufacturing systems: theory and practice, Springer.

Clerc, M. 2004. Discrete particle swarm optimization, illustrated by the traveling salesman problem. New Optimization Techniques in Engineering. Springer Berlin Heidelberg.

Clerc, M. & Kennedy, J. 2002. The particle swarm-explosion, stability, and convergence in a multidimensional complex space. IEEE Transactions on Evolutionary Computation, 6, 58-73.

Dai, M., Tang, D., Giret, A., Salido, M. A. & Li, W. 2013. Energy-efficient scheduling for a flexible flow shop using an improved genetic-simulated annealing algorithm. Robotics and Computer-Integrated Manufacturing, 29, 418-429.

Daoud, S., Chehade, H., Yalaoui, F. & Amodeo, L. 2014. Solving a robotic assembly line balancing problem using efficient hybrid methods. Journal of Heuristics, 20, 235-259.

Dar-El, E. 1973. MALB—A heuristic technique for balancing large single-model assembly lines. AIIE transactions, 5, 343-356.

Dar-El, E. & Rubinovitch, Y. 1979. MUST-A multiple solutions technique for balancing single model assembly lines. Management Science, 25, 1105-1114.

Davis, L. Applying adaptive algorithms to epistatic domains. 1985. In: Proceedings of IJCAI, 162-164.

Deckro, R. F. & Rangachari, S. 1990. A goal approach to assembly line balancing. Computers & Operations Research, 17, 509-521.

Domm, R. W. 2009. Michigan Yesterday & Today. Voyageur Press.

Eberhart, R. C. & Shi, Y. Comparison between genetic algorithms and particle swarm optimization. In: Proceedings of Evolutionary Programming VII, 1998. Springer, 611-616.

Eberhart, R. C. & Shi, Y. 2000. Comparing inertia weights and constriction factors in particle swarm optimization. In: Proceedings of Evolutionary Computation, IEEE, 84-88.

Economics, Z. 2011. Simple assembly line balancing using particle swarm optimization algorithm. International Journal of Digital Content Technology and its Applications, 5, 297-304.

El-Dib, A., Youssef, H. K., El-Metwally, M. & Osman, Z. 2004. Load flow solution using hybrid particle swarm optimization. In: Proceedings of International Conference on Electrical, Electronic and Computer Engineering, IEEE, 742-746.

Erel, E., Sabuncuoglu, I. & Aksu, B. 2001. Balancing of U-type assembly systems using simulated annealing. International Journal of Production Research, 39, 3003-3015.

Erel, E., Sabuncuoglu, I. & Sekerci, H. 2005. Stochastic assembly line balancing using beam search. International Journal of Production Research, 43, 1411-1426.

Erel, E. & Sarin, S. C. 1998. A survey of the assembly line balancing procedures. Production Planning & Control, 9, 414-434.

Faaland, B. H., Klastorin, T. D., Schmitt, T. G. & Shtub, A. 1992. Assembly Line Balancing with Resource Dependent Task Times. Decision Sciences, 23, 343-364.

Feoktistov, V. 2006. Differential evolution, Springer.

Fernandes, L. 1992. Heuristic methods for the mixed-model assembly line balancing and sequencing. MSc. Dissertation, Lehigh University.

Fetters-Walp,E.2010. Moving to a quicker pace Available: http://www.boeing.com/Features/2010/05/bca_moving_line_05_24_10.html.

Fysikopoulos, A., Anagnostakis, D., Salonitis, K. & Chryssolouris, G. 2012. An Empirical Study of the Energy Consumption in Automotive Assembly. Procedia CIRP, 3, 477-482.

Gao, J., Sun, L., Wang, L. & Gen, M. 2009. An efficient approach for type II robotic assembly line balancing problems. Computers & Industrial Engineering, 56, 1065-1080.

Gen, M., Cheng, R. & Lin, L. 2008. Assembly Line Balancing Models. Network Models and Optimization: Multiobjective Genetic Algorithm Approach, 477-550.

Ghosh, S. & Gagnon, R. J. 1989. A comprehensive literature review and analysis of the design, balancing and scheduling of assembly systems. International Journal of Production Research, 27, 637-670.

Glover, F., & Kochenberger, G. A. (Eds.). 2003) Handbook of metaheuristics. Springer Science & Business Media.

Gökçen, H. & Agˇ Pak, K. 2006. A goal programming approach to simple U-line balancing problem. European Journal of Operational Research, 171, 577-585.

Gökçen, H., Ağpak, K. & Benzer, R. 2006. Balancing of parallel assembly lines. International Journal of Production Economics, 103, 600-609.

Gonçalves, J. F. & De Almeida, J. R. 2002. A hybrid genetic algorithm for assembly line balancing. Journal of Heuristics, 8, 629-642.

Graves, S. C. & Lamar, B. W. 1983. An integer programming procedure for assembly system design problems. Operations Research, 31, 522-545.

Graves, S. C. & Redfield, C. H. 1988. Equipment selection and task assignment for multiproduct assembly system design. International Journal of Flexible Manufacturing Systems, 1, 31-50.

Gungor, A. & Gupta, S. M. 1999. Issues in environmentally conscious manufacturing and product recovery: a survey. Computers & Industrial Engineering, 36, 811-853.

Gunther, R. E., Johnson, G. D. & Peterson, R. S. 1983. Currently practiced formulations for the assembly line balance problem. Journal of Operations Management, 3, 209-221.

Guo, W., Li, W., Zhang, Q., Wang, L., Wu, Q. & Ren, H. 2013. Biogeography-based particle swarm optimization with fuzzy elitism and its applications to constrained engineering problems. Engineering Optimization, 46, 1465-1484.

Gutjahr, A. L. & Nemhauser, G. L. 1964. An algorithm for the line balancing problem. Management Science, 11, 308-315.

Hahn, R. 1972. Produktionsplanung bei Linienfertigung, .

Hamta, N., Fatemi Ghomi, S. M. T., Jolai, F., & Akbarpour Shirazi, M. 2013. A hybrid PSO algorithm for a multi-objective assembly line balancing problem with flexible operation times, sequence-dependent setup times and learning effect. International Journal of Production Economics, 141(1), 99-111.

Hazır, O., Delorme, X. & Dolgui, A. 2014. A Survey on Cost and Profit Oriented Assembly Line Balancing. In: Proceedings of The International Federation of Automatic Control, Cape Town, South Africa. 6159-6167.

He, Y., Liu, B., Zhang, X., Gao, H. & Liu, X. 2012. A modeling method of task-oriented energy consumption for machining manufacturing system. Journal of Cleaner Production, 23, 167-174.

Held, M., Karp, R. M. & Shareshian, R. 1963. Assembly-line balancing-dynamic programming with precedence constraints. Operations Research, 11, 442-459.

Helgeson, W. & Birnie, D. 1961. Assembly line balancing using the ranked positional weight technique. Journal of Industrial Engineering, 12, 394-398.

Hirano, H. 1988. JIT factory revolution: A pictorial guide to factory design of the future, Productivity Press.

Hoffmann, T. R. 1992. EUREKA: A hybrid system for assembly line balancing. Management Science, 38, 39-47.

Hu, X., Zhang, Y., Zeng, N. & Wang, D. 2014. A Novel Assembly Line Balancing Method Based on PSO Algorithm. Mathematical Problems in Engineering, Volume(2014), Article ID 743695, Pages 10.

Huang, K.-W., Chen, J.-L., Yang, C.-S. & Tsai, C.-W. 2014. A memetic particle swarm optimization algorithm for solving the DNA fragment assembly problem. Neural Computing and Applications, 1-12. doi:10.1007/s00521-014-1659-0.

Hwang, R. K., Katayama, H. & Gen, M. 2008. U-shaped assembly line balancing problem with genetic algorithm. International Journal of Production Research, 46, 4637-4649.

Jackson, J. R. 1956. A computing procedure for a line balancing problem. Management Science, 2, 261-271.

Johnson, R. V. 1988. Optimally balancing large assembly lines with "FABLE". Management Science, 34, 240-253.

Kaelo, P. & Ali, M. 2006. A numerical study of some modified differential evolution algorithms. European journal of operational research, 169, 1176-1184.

Kao, E. P. & Queyranne, M. 1982. On dynamic programming methods for assembly line balancing. Operations Research, 30, 375-390.

Kara, Y. 2008. Line balancing and model sequencing to reduce work overload in mixed-model U-line production environments. Engineering Optimization, 40, 669-684.

Karp, R. M. 1972. Reducibility among combinatorial problems, Springer.

Kennedy, J. & Eberhart, R. 1995. Particle swarm optimization. In: Proceedings of International Conference on Neural Networks, IEEE, 1942-1948 vol.4.

Khaw, C. L. & Ponnambalam, S. G. 2009. Multi-rule multi-objective ant colony optimization for straight and U-type assembly line balancing problem. In: Proceedings of International Conference on Automation Science and Engineering, IEEE, 177-182.

Khouja, M., Booth, D. E., Suh, M. & Mahaney Jr, J. K. 2000. Statistical procedures for task assignment and robot selection in assembly cells. International Journal of Computer Integrated Manufacturing, 13, 95-106.

Kilian, L. 2008. The economic effects of energy price shocks. Journal of Economic Literature, 871-909.

Kilincci, O. 2010. A Petri net-based heuristic for simple assembly line balancing problem of type 2. The International Journal of Advanced Manufacturing Technology, 46, 329-338.

Kilincci, O. & Bayhan, G. M. 2006. A Petri net approach for simple assembly line balancing problems. The International Journal of Advanced Manufacturing Technology, 30, 1165-1173.

Kim, H. & Park, S. 1995. A strong cutting plane algorithm for the robotic assembly line balancing problem. International Journal of Production Research, 33, 2311-2323.

Kim, Y. K., Kim, J. Y. & Kim, Y. 2006. An endosymbiotic evolutionary algorithm for the integration of balancing and sequencing in mixed-model U-lines. European Journal of Operational Research, 168, 838-852.

Kim, Y. K., Kim, S. J. & Kim, J. Y. 2000. Balancing and sequencing mixed-model U-lines with a co-evolutionary algorithm. Production Planning & Control, 11, 754-764.

Kim, Y. K. & Kim, Y. J. 1996. Genetic algorithms for assembly line balancing with various objectives. Computers & Industrial Engineering, 30, 397-409.

Klein, R. & Scholl, A. 1996. Maximizing the production rate in simple assembly line balancing—a branch and bound procedure. European Journal of Operational Research, 91, 367-385.

Kriengkorakot, N. & Pianthong, N. 2012. The assembly line balancing problem: review articles. KKU Engineering Journal, 34, 133-140.

Kubota, Y. 2011. Toyota's New Assembly Line Saves Time, Costs: Report. Available:http://www.reuters.com/assets/print?aid=USTRE70M07H20110123

Kumar, D. M. 2013. Assembly Line Balancing: A Review of Developments and Trends in Approach to Industrial Application. Global Journal of Researches In Engineering, 13.

Lai, L. K. & Liu, J. N. 2009. ALBO: An assembly line balance optimization model using ant colony optimization. In: Proceedings of International Conference on Natural Computation, IEEE, 8-12.

Lee, K. Y. & Park, J.-B. 2006. Application of particle swarm optimization to economic dispatch problem: advantages and disadvantages. In: Proceedings of IEEE Power Systems Conference and Exposition, PSCE'06, IEEE, 188-192.

Levitin, G., Rubinovitz, J. & Shnits, B. 2006. A genetic algorithm for robotic assembly line balancing. European Journal of Operational Research, 168, 811-825.

Liu, Y., Dong, H., Lohse, N., Petrovic, S. & Gindy, N. 2014. An investigation into minimising total energy consumption and total weighted tardiness in job shops. Journal of Cleaner Production, 65, 87-96.

Long, W., Liang, X., Huang, Y. & Chen, Y. 2014. An effective hybrid cuckoo search algorithm for constrained global optimization. Neural Computing and Applications, 25(3–4), 911–926.

Luo, H., Du, B., Huang, G. Q., Chen, H. & Li, X. 2013. Hybrid flow shop scheduling considering machine electricity consumption cost. International Journal of Production Economics, 146, 423-439.

Lutz, L. 1974. Abtakten von Montagelinien, Krausskopf.

Mansoor, E. M. & Yadin, M. 1971. On the problem of assembly line balancing, in Developments in Operations Research, Avi-Itzhak, B. (ed), Gordon and Breach, New York, p.361.

Marinakis, Y. & Marinaki, M. 2010. A hybrid genetic–Particle Swarm Optimization Algorithm for the vehicle routing problem. Expert Systems with Applications, 37, 1446-1455.

Martí, R. & Reinelt, G. 2011. The linear ordering problem: exact and heuristic methods in combinatorial optimization, Springer.

Martinez, U. & Duff, W. S. 2004. Heuristic approaches to solve the U-shaped line balancing problem augmented by genetic algorithms. In: Proceedings of Systems and Information Engineering Design Symposium, IEEE, 287-293.

Miltenburg, G. & Wijngaard, J. 1994. The U-line line balancing problem. Management Science, 40, 1378-1388.

Miltenburg, J. 1998. Balancing U-lines in a multiple U-line facility. European Journal of Operational Research, 109, 1-23.

Miltenburg, J. & Sparling, D. 1995. Optimal solution algorithms for the U-line balancing problem. Working Paper, McMaster University, Hamilton, Ontario, Canada.

Mohamed, A. W., Sabry, H. Z. & Khorshid, M. 2012. An alternative differential evolution algorithm for global optimization. Journal of Advanced Research, 3, 149-165.

Monden, Y. 1983. Toyota production system: practical approach to production management, Industrial Engineering and Management Press, Institute of Industrial Engineers Norcross, GA.

Mouzon, G. & Yildirim, M. B. 2008. A framework to minimise total energy consumption and total tardiness on a single machine. International Journal of Sustainable Engineering, 1, 105-116.

Mouzon, G., Yildirim, M. B. & Twomey, J. 2007. Operational methods for minimization of energy consumption of manufacturing equipment. International Journal of Production Research, 45, 4247-4271.

Naka, S., Genji, T., Yura, T. & Fukuyama, Y. 2003. A hybrid particle swarm optimization for distribution state estimation. IEEE Transactions on Power Systems, 18, 60-68.

Nakade, K. & Ohno, K. 1999. An optimal worker allocation problem for a U-shaped production line. International Journal of Production Economics, 60, 353-358.

Nearchou, A. C. 2005. A differential evolution algorithm for simple assembly line balancing. In: Proceedings of 16th International Federation of Automatic Control (IFAC) World Congress, Prague, 1462-1462.

Nearchou, A. C. 2006. Meta-heuristics from nature for the loop layout design problem. International Journal of Production Economics, 101, 312-328.

Nearchou, A. C. 2007. Balancing large assembly lines by a new heuristic based on differential evolution method. The International Journal of Advanced Manufacturing Technology, 34, 1016-1029.

Nearchou, A. C. & Omirou, S. L. 2006. Differential evolution for sequencing and scheduling optimization. Journal of Heuristics, 12, 395-411.

Ngai, E. W., Ng, C. D. & Huang, G. 2013. Energy Sustainability for Production Design and Operations. International Journal of Production Economics, 146, 383-385.

Nicosia, G., Pacciarelli, D. & Pacifici, A. 2002. Optimally balancing assembly lines with different workstations. Discrete Applied Mathematics, 118, 99-113.

Owen, T. 1986. Assembly with robots, Prentice-Hall, Inc.

Padrón, M., Irizarry, M. D. L. A., Resto, P. & Mejía, H. P. 2009. A methodology for cost-oriented assembly line balancing problems. Journal of Manufacturing Technology Management, 20, 1147-1165.

Pan, C. 2005. Integrating CAD files and automatic assembly sequence planning. Phd Dissertation. Iowa State University.

Petropoulos, D. I. & Nearchou, A. C. 2011. A particle swarm optimization algorithm for balancing assembly lines. Assembly Automation, 31, 118-129.

Pierreval, H., Caux, C., Paris, J. & Viguier, F. 2003. Evolutionary approaches to the design and organization of manufacturing systems. Computers & Industrial Engineering, 44, 339-364.

Plans, J. & Corominas, A. 1999. Modelling and solving the SALB-E problem. In: Proceedings of International Symposium on Assembly and Task Planning, IEEE, 356-360.

Ponnambalam, S., Aravindan, P. & Naidu, G. M. 1999. A comparative evaluation of assembly line balancing heuristics. The International Journal of Advanced Manufacturing Technology, 15, 577-586.

Ponnambalam, S., Aravindan, P. & Naidu, G. M. 2000. A multi-objective genetic algorithm for solving assembly line balancing problem. The International Journal of Advanced Manufacturing Technology, 16, 341-352.

Qin, A. K. & Suganthan, P. N. 2005. Self-adaptive differential evolution algorithm for numerical optimization. In: Proceedings of Congress on Evolutionary Computation, IEEE, 1785-1791.

Rao, S. S. 2009. Engineering optimization: theory and practice. John Wiley & Sons.

Rameshkumar, K., Suresh, R. K., & Mohanasundaram, K. M. (2005). Discrete particle swarm optimization (DPSO) algorithm for permutation flowshop scheduling to minimize makespan. In: Proceedings of Advances in Natural Computation, Springer Berlin Heidelberg ,572-581.

Rashid, M. F. F., Hutabarat, W. & Tiwari, A. 2012. A review on assembly sequence planning and assembly line balancing optimisation using soft computing approaches. The International Journal of Advanced Manufacturing Technology, 59, 335-349.

Ratnaweera, A., Halgamuge, S. & Watson, H. C. 2004. Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. IEEE Transactions on Evolutionary Computation, 8, 240-255.

Rosenberg, O. & Ziegler, H. 1992. A comparison of heuristic algorithms for cost-oriented assembly line balancing. Zeitschrift für Operations Research, 36, 477-495.

Roshani, A., Fattahi, P., Roshani, A., Salehi, M. & Roshani, A. 2012. Cost-oriented two-sided assembly line balancing problem: A simulated annealing approach. International Journal of Computer Integrated Manufacturing, 25, 689-715.

Rothlauf, F. 2011. Design of modern heuristics: principles and application, Springer.

Rubinovitz, J. & Bukchin, J. 1991. Design and balancing of robotic assembly lines, Society of Manufacturing Engineers.

Rubinovitz, J., Bukchin, J. & Lenz, E. 1993. RALB–A heuristic algorithm for design and balancing of robotic assembly lines. CIRP Annals-Manufacturing Technology, 42, 497-500.

Rubinovitz, J. & Levitin, G. 1995. Genetic algorithm for assembly line balancing. International Journal of Production Economics, 41, 343-354.

Salveson, M. E. 1955. The assembly line balancing problem. Journal of Industrial Engineering, 6, 18-25.

Sanderson, A. C., De Mello, L. S. H. & Zhang, H. 1990. Assembly sequence planning. AI Magazine, 11, 62.

Sarin, S. C., Erel, E. & Dar-El, E. M. 1999. A methodology for solving single-model, stochastic assembly line balancing problem. Omega, 27, 525-535.

Scholl, A. 1995. Data of assembly line balancing problems. Darmstadt Technical University, Department of Business Administration, Economics and Law, Institute for Business Studies (BWL).

Scholl, A. 1999. Balancing and sequencing of assembly lines, Physica-Verlag Heidelberg.

Scholl, A. & Becker, C. 2005. A note on "An exact method for cost-oriented assembly line balancing". International Journal of Production Economics, 97, 343-352.

Scholl, A. & Becker, C. 2006. State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. European Journal of Operational Research, 168, 666-693.

Scholl, A., Boysen, N. & Fliedner, M. 2008. The sequence-dependent assembly line balancing problem. OR Spectrum, 30, 579-609.

Scholl, A. & Klein, R. 1997. SALOME: A bidirectional branch-and-bound procedure for assembly line balancing. INFORMS Journal on Computing, 9, 319-334.

Scholl, A. & Klein, R. 1999a. Balancing assembly lines effectively–a computational comparison. European Journal of Operational Research, 114, 50-58.

Scholl, A. & Klein, R. 1999b. ULINO: Optimally balancing U-shaped JIT assembly lines. International Journal of Production Research, 37, 721-736.

Scholl, A. & Voß, S. 1997. Simple assembly line balancing—Heuristic approaches. Journal of Heuristics, 2, 217-244.

Schrage, L. & Baker, K. R. 1978. Dynamic programming solution of sequencing problems with precedence constraints. Operations research, 26, 444-449.

Seyed-Alagheband, S., Ghomi, S. F. & Zandieh, M. 2011. A simulated annealing algorithm for balancing the assembly line type II problem with sequence-dependent setup times between tasks. International Journal of Production Research, 49, 805-825.

Shi, Y. & Eberhart, R. C. 1998. Parameter selection in particle swarm optimization. In: Proceedings of Evolutionary Programming VII, Springer, 591-600.

Shrouf, F., Ordieres-Meré, J., García-Sánchez, A. & Ortega-Mier, M. 2014. Optimizing the production scheduling of a single machine to minimize total energy consumption costs. Journal of Cleaner Production, 67, 197-207.

Sirovetnukul, R. & Chutima, P. 2010. Multi-objective particle swarm optimization with negative knowledge for U-shaped assembly line worker allocation problems.

In: Proceedings of International Conference on Industrial Engineering and Engineering Management (IEEM), IEEE, 2033-2038.

Sivanandam, S. N., Visalakshi, P., & Bhuvaneswari, A. 2007. Multiprocessor Scheduling Using Hybrid Particle Swarm Optimization with Dynamically Varying Inertia. International Journal of Computer Science & Applications, 4(3), 95-106.

Sivasankaran, P. & Shahabudeen, P. 2014. Literature review of assembly line balancing problems. The International Journal of Advanced Manufacturing Technology, 1-30.

Sörensen, K. & Glover, F. W. 2013. Metaheuristics. Encyclopedia of Operations Research and Management Science. Springer.

Storn, R. & Price, K. 1997. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. Journal of global optimization, 11, 341-359.

Suresh, G. & Sahu, S. 1994. Stochastic assembly line balancing using simulated annealing. International Journal of Prodction Research, 32, 1801-1810.

Thangavelu, S. & Shetty, C. 1971. Assembly line balancing by zero-one integer programming. AIIE Transactions, 3, 61-68.

Ting, T. O., Yang, X. S., Cheng, S., & Huang, K. 2015. Hybrid Metaheuristic Algorithms: Past, Present, and Future. In: Proceedings of Recent Advances in Swarm Intelligence and Evolutionary Computation (Springer International Publishing), 71-83.

Toklu, B. & Özcan, U. 2008. A fuzzy goal programming model for the simple U-line balancing problem with multiple objectives. Engineering Optimization, 40, 191-204.

Toksarı, M. D., İşleyen, S. K., Güner, E. & Baykoç, Ö. F. 2008. Simple and U-type assembly line balancing problems with a learning effect. Applied Mathematical Modelling, 32, 2954-2961.

Tonge, F. M. 1960. A heuristic program for assembly line balancing. Mathematics Division, RAND Corporation, Santa Monica. California, U.S.A.

Törenli, A. 2009. Assembly line design and optimization. MSc. Dissertation, Chalmers University of Technology.

Tsai, D.-M. & Yao, M.-J. 1993. A line-balance-based capacity planning procedure for series-type robotic assembly line. International Journal of Production Research, 31, 1901-1920.

Urban, T. L. 1998. Note. Optimal balancing of U-shaped assembly lines. Management Science, 44, 738-741.

Valian, E., Mohanna, S. & Tavakoli, S. 2011. Improved cuckoo search algorithm for global optimization. International Journal of Communications and Information Technology, 1, 31-44.

Wei, N.-C. & Chao, I. 2011. A solution procedure for type E simple assembly line balancing problem. Computers & Industrial Engineering, 61, 824-830.

Wu, E.-F., Jin, Y., Bao, J.-S. & Hu, X.-F. 2008. A branch-and-bound algorithm for two-sided assembly line balancing. The International Journal of Advanced Manufacturing Technology, 39, 1009-1015.

Wu, Y. C., Lee, W. P., & Chien, C. W. 2011. Modified the performance of differential evolution algorithm with dual evolution strategy. In: Proceedings of 2009 International Conference on Machine Learning and Computing, Vol. 3, 57-63.

Yalaoui, A., Chehade, H., Yalaoui, F. & Amodeo, L. 2013. Optimization of Logistics, John Wiley & Sons.

Yang, X.-S. & Deb, S. Cuckoo search via Lévy flights. 2009. In: Proceedings of World Congress on Nature & Biologically Inspired Computing, IEEE, 210-214.

Yang, X.-S. & Deb, S. 2014. Cuckoo search: recent advances and applications. Neural Computing and Applications, 24, 169-174.

Yazdanparast, V., Hajihosseini, H. & Bahalke, A. 2011. Cost Oriented Assembly Line Balancing Problem with Sequence Dependent Setup Times. Journal of Applied Sciences Research, 7.

Yoosefelahi, A., Aminnayeri, M., Mosadegh, H. & Ardakani, H. D. 2012. Type II robotic assembly line balancing problem: An evolution strategies algorithm for a multi-objective model. Journal of Manufacturing Systems, 31, 139-151.

Zhang, Z. & Cheng, W. An Exact Method for U-shaped Assembly Line Balancing Problem. In: Proceedings of International Workshop on Intelligent Systems and Applications (ISA), 2010. IEEE, 1-4.

Zheng, Q., Li, M., Li, Y. & Tang, Q. 2013. Station ant colony optimization for the type 2 assembly line balancing problem. The International Journal of Advanced Manufacturing Technology, 66, 1859-1870.

APPENDIX

Appendix 1- Performance times of 35 tasks by 5 Robots

Task	Predecessor Tasks	Robot 1	Robot 2	Robot 3	Robot 4	Robot 5
1	-	142	67	88	56	84
2	1	92	45	183	56	69
3	2	56	25	36	37	37
4	3	64	61	53	45	47
5	1	62	29	92	35	95
6	5	68	51	132	83	177
7	1,6	93	90	137	71	158
8	6	59	73	90	51	116
9	8	29	36	36	51	50
10	1	53	55	37	36	43
11	4	63	40	85	59	51
12	1	42	73	49	109	49
13	9	42	36	91	64	47
14	7,10	77	46	93	68	66
15	14	37	45	50	37	83
16	15	28	28	73	47	37
17	-	65	49	41	53	51
18	7,12	93	49	63	151	101
19	18	103	37	62	54	89
20	17,19	38	55	38	29	34
21	16,20	51	83	122	67	117
22	21	36	43	57	47	60
23	22	70	87	74	63	146
24	23	42	83	108	49	49
25	21	103	55	66	54	83
26	25	36	78	64	34	48
27	24,26	44	82	49	68	46
28	11,13	105	36	69	119	94
29	28	58	31	59	37	60
30	21	43	37	59	53	64
31	30	42	32	51	82	113
32	21,31	40	39	89	45	91
33	11,13,27,32	32	31	99	57	46
34	27	93	37	50	53	91
35	33	37	50	72	84	53

Appendix 2- Power data for small size datasets (Power in kW)

Problem	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15	R16	R17	R18	R19
11-4	0.25	0.4	0.3	0.35	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
25-3	0.4	0.35	0.3	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
25-4	0.25	0.4	0.3	0.3	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
25-6	0.3	0.4	0.4	0.3	0.3	0.35	--	--	--	--	--	--	--	--	--	--	--	--	--
25-9	0.2	0.3	0.25	0.4	0.35	0.4	0.25	0.3	0.3	--	--	--	--	--	--	--	--	--	--
35-4	0.5	0.8	0.8	0.9	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
35-5	0.7	0.5	0.8	0.9	0.5	--	--	--	--	--	--	--	--	--	--	--	--	--	--
35-7	0.9	0.8	0.5	0.9	0.7	0.8	0.5	--	--	--	--	--	--	--	--	--	--	--	--
35-12	0.9	0.8	0.6	0.8	0.9	0.5	0.5	0.6	0.7	0.5	0.7	0.9	--	--	--	--	--	--	--
53-5	1.1	1.2	1.5	0.9	1.3	--	--	--	--	--	--	--	--	--	--	--	--	--	--
53-7	1.5	0.9	1.2	1.1	1.2	1.3	1.5	--	--	--	--	--	--	--	--	--	--	--	--
53-10	1.1	0.9	1.2	1.5	1.3	0.9	1.2	1.4	1.1	0.9	--	--	--	--	--	--	--	--	--
53-14	0.9	1.2	1.5	1.1	1.4	0.9	1.2	0.9	1.5	1.3	1.2	1.4	1.3	0.9	--	--	--	--	--
70-7	1.4	1.4	1.7	1.5	1.3	1.5	1.4	--	--	--	--	--	--	--	--	--	--	--	--
70-10	1.2	1.6	1.1	1.7	1.2	1.5	1.4	1.8	1.6	1.7	--	--	--	--	--	--	--	--	--
70-14	1.8	1.4	1.5	1.7	1.1	1.4	1.5	1.3	1.8	1.6	1.1	1.7	1.4	1.2	--	--	--	--	--
70-19	1.5	1.8	1.1	1.7	1.3	1.6	1.6	1.2	1.8	1.3	1.7	1.6	1.3	1.1	1.3	1.1	1.1	1.5	1.7

*Source of the data: Randomly generated.

Appendix 3 -Robot Power Details for Large Size data (Power in kW)

Problem	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15	R16	R17	R18	R19	R20	R21	R22	R23	R24	R25
89-8	1.1	1.8	1.5	1.2	1.9	1.4	1.2	1.6	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
89-12	1.8	1.2	1.4	1.2	1.4	1.4	1.6	1.7	1.9	1.5	1.1	1.5	--	--	--	--	--	--	--	--	--	--	--	--	--
89-16	1.5	1.6	1.7	1.4	1.6	1.7	1.4	1.5	1.8	1.4	1.2	1.5	1.6	1.4	1.3	1.3	--	--	--	--	--	--	--	--	--
89-21	1.5	1.1	1.7	1.8	1.6	1.4	1.7	1.8	1.3	1.9	1.9	1.5	1.4	1.3	1.6	1.6	1.4	1.9	1.6	1.1	1.4	--	--	--	--
111-9	2.1	1.7	2.4	1.8	1.2	2.2	1.5	1.8	2.3	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
111-13	1.8	2.1	1.5	2.2	2.3	2.4	1.9	1.7	1.9	2.2	2.1	2.3	2.4	--	--	--	--	--	--	--	--	--	--	--	--
111-17	1.7	1.9	1.6	2.2	2.1	1.8	1.6	1.5	2.1	1.6	2.3	1.9	1.8	2.4	1.7	2.2	2.1	--	--	--	--	--	--	--	--
111-22	2.1	1.5	1.5	1.6	1.7	1.6	2.1	2.4	1.8	1.9	1.5	1.7	1.6	2.2	1.8	1.6	2.3	2.1	1.6	1.7	2.2	2.4	--	--	--
148-10	2.1	2.2	1.8	2.4	1.6	1.5	2.0	2.3	2.4	2.5	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
148-14	2.2	2.0	2.1	2.4	1.8	2.4	1.7	1.5	1.7	2.1	2.2	2.0	1.7	2.3	--	--	--	--	--	--	--	--	--	--	--
148-21	2.4	1.5	1.6	2.4	2.4	1.5	1.9	1.6	2.0	2.0	2.1	2.2	2.0	2.1	1.5	1.9	2.0	2.1	1.8	2.0	2.0	--	--	--	--
148-29:	2.2	2.3	1.6	1.6	2.4	2.3	2.0	1.6	2.2	1.6	1.6	2.1	1.7	1.6	1.7	1.7	2.4	2.2	2.2	1.8	1.8	1.5	2.1	1.9	1.8
297-19	2.8	2.2	2.3	2.4	2.6	2.4	1.9	2.0	2.2	2.3	1.9	2.6	3.0	2.7	2.5	2.5	2.2	2.1	2.7	--	--	--	--	--	--
297-29:	2.3	1.9	2.3	2.1	2.1	2.0	2.9	2.1	2.2	2.8	2.7	2.5	2.1	2.8	2.0	0.1	3.0	2.4	2.7	2.9	2.2	2.2	2.1	2.1	2.8
297-38:	2.3	2.3	1.8	2.0	2.9	1.8	2.4	2.3	2.9	2.0	2.8	2.0	2.7	2.9	2.7	2.2	2.3	2.5	2.6	2.5	2.1	1.8	2.1	2.9	2.5
297-50:	2.4	2.3	2.5	2.4	2.2	2.0	2.5	2.3	2.3	2.4	2.0	2.7	2.3	2.6	2.4	2.4	2.5	2.1	2.3	1.9	2.0	2.3	2.9	2.2	2.4

Problem	R26	R27	R28	R29	R30	R31	R32	R33	R34	R35	R36	R37	R38	R39	R40	R41	R42	R43	R44	R45	R46	R47	R48	R49	R50
89-8	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
89-12	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
89-16	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
89-21	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
111-9	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
111-13	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
111-17	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
111-22	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
148-10	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
148-14	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
148-21	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
148-29:	1.9	2.4	1.7	1.5	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
297-19	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
297-29:	2.1	2.7	2.6	2.7	2.9	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
297-38:	2.8	1.9	1.9	2.1	2.7	2.4	2.2	1.9	2.6	2.7	3.0	2.5	2.8	--	--	--	--	--	--	--	--	--	--	--	--
297-50:	2.2	2.7	1.8	2.9	1.8	1.8	1.8	2.4	2.3	2.2	2.2	1.9	1.8	2.6	2.1	2.6	2.5	2.6	2.5	1.8	2.8	2.3	2.4	2.9	2.7

*Source of the data: Randomly generated.

Appendix 4 -Robot Cost data for small size datasets

Problem	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15	R16	R17	R18	R19
11-4	1.1	1.2	1.25	1.3	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
25-3	1	1.5	1.2	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
25-4	1	1.25	1.15	1.2	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
25-6	1.05	0.95	1	1.2	1.5	1.3	--	--	--	--	--	--	--	--	--	--	--	--	--
25-9	1.3	1.5	1	1.2	0.95	1	1.1	1.25	1.1	--	--	--	--	--	--	--	--	--	--
35-4	1.05	0.95	1	1.2		--	--	--	--	--	--	--	--	--	--	--	--	--	--
35-5	1	1.5	0.8	1.2	0.87	--	--	--	--	--	--	--	--	--	--	--	--	--	--
35-7	1.35	0.95	1.1	1.25	1	1.5	1.15	--	--	--	--	--	--	--	--	--	--	--	--
35-12	1.15	1.25	0.825	0.95	1	1.5	1.35	1.1	1.2	0.875	1.15	0.975	--	--	--	--	--	--	
53-5	1	1.25	0.95	1.2	1.15	--	--	--	--	--	--	--	--	--	--	--	--	--	--
53-7	1.2	1.25	1.025	0.95	1.1	1.3	1.15						--	--	--	--	--	--	--
53-10	1.3	1.05	1.1	1.35	1.15	1.25	1.2	1.225	1.4	0.95			--	--	--	--	--	--	--
53-14	1.2	1.25	1.025	0.95	1.1	1.3	1.2	1.35	1.4	0.925	0.9	1.05	1.15	1	--	--	--	--	--
70-7	1	1.3	1.15	1.05	1.1	1.25	1.2					--	--	--	--	--	--	--	--
70-10	1.3	1.05	1.1	1.35	1.15	1.25	1.2	1.225	1.4	0.95	--	--	--	--	--	--	--	--	--
70-14	1.2	1.25	1.025	0.95	1.1	1.3	1.2	1.35	1.4	0.925	0.9	1.05	1.15	1					
70-19	1	0.82	0.9	1.05	1.3	1.4	1	1.1	0.95	1.225	0.95	1.2	1.35	1.25	1.325	1.15	1.25	1.3	0.8

*All the values are to be multiplied by 10^6 to get the actual data

Appendix 5 -Robot Cost data for large size datasets

Problem	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15	R16	R17	R18	R19	R20	R21	R22	R23	R24	R25
89-8	1	1.3	1.15	1.05	0.85	1.25	1.2	1.1	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
89-12	1.15	1.25	0.825	0.95	1	1.5	1.35	1.1	1.2	0.875	1.15	0.975	--	--	--	--	--	--	--	--	--	--	--	--	--
89-16	1.225	1.325	1.3	0.9	1.1	1.25	1.1	1.15	1	1.05	1.3	0.95	1.05	1.4	1.325	1.225	--	--	--	--	--	--	--	--	--
89-21	1.225	1.325	1.3	0.8	0.95	1.25	1.1	1.15	1	1.05	1.3	0.95	1.05	0.85	1.325	1.225	0.95	0.9	1.4	1.2	1.1	--	--	--	--
111-9	1.35	1.2	1.3	1.05	0.95	1.25	1.1	1.15	1.2	--	--	--	1.35	--	--	--	--	--	--	--	--	--	--	--	--
111-13	1.05	1.325	1.3	1.15	1.225	1.25	1.1	1.15	1	1.225	1.2	0.95	0.9	1.35	1	1.225	1.05	--	--	--	--	--	--	--	--
111-17	1.05	1.325	1.3	1.15	1.225	1.25	1.1	1.15	1	1.225	1.2	0.95	1	0.8	1.3	1.2	0.95	1.2	1.3	1.1	1.2	0.9	--	--	--
111-22	1	1.4	1.3	0.8	0.95	1.25	1	1.1	1	1	1.3	0.95	1	0.8	1.3	1.2	0.95	1.2	1.3	1.1	1.2	0.9	--	--	--
148-10	1.3	1.325	0.95	1.2	1.4	1.25	1.225	1.15	1	1.05	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
148-14	1.2	1.25	1	0.95	1	1.3	1.2	1.3	1.4	0.9	0.9	1	1.15	1	--	--	--	--	--	--	--	--	--	--	--
148-21	1.225	1.325	1.3	0.8	0.95	1.25	1.1	1.15	1	1.05	1.3	0.95	1.05	0.85	1.32	1.225	0.95	0.9	1.4	1.2	1.1	--	--	--	--
148-29:	1.225	1.325	1.3	0.8	0.95	1.25	1.25	1.1	1.15	1	1.05	1.3	0.95	1.05	0.85	1.325	1.225	0.95	1.4	1.2	1.1	1.225	1.325	1.3	0.8
297-19	1.225	1.325	1.3	0.8	0.95	1.25	1.1	1.15	1	1.05	1.3	0.95	1.05	0.85	1.325	1.225	0.95	0.9	1.4	--	--	--	--	--	--
297-29:	1.2	1.3	1.3	0.8	0.95	1.25	1	1.15	1	1	1.3	0.95	1	0.85	1.3	1.2	0.95	0.9	1.4	1.2	1.1	1.2	1.3	1.3	0.8
297-38:	1.2	1.3	1.3	1.05	1.3	1.25	1	1.1	1.1	1	1.3	1	1	0.9	1.3	1.2	0.9	1	1.4	1.2	1.1	1.2	1.3	1.3	0.8
297-50:	1.225	1.225	1.325	1.3	1.15	0.95	1.25	1.1	1.15	1	1.05	1.3	0.95	1.05	0.85	1.325	1.325	1.255	0.95	0.9	1.32	1.3	1.05	1.35	1.25

Problem	R26	R27	R28	R29	R30	R31	R32	R33	R34	R35	R36	R37	R38	R39	R40	R41	R42	R43	R44	R45	R46	R47	R48	R49	R50
89-8	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
89-12	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
89-16	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
89-21	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
111-9	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
111-13	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
111-17	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
111-22	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
148-10	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
148-14	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
148-21	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
148-29:	0.95	1.25	1.1	1.15	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
297-19	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
297-29:	0.95	1.25	1	1.1	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
297-38:	0.95	1.25	1.1	1.15	1.2	1.3	1.3	1.15	0.95	1.25	1	1.15	1	--	--	--	--	--	--	--	--	--	--	--	--
297-50:	1.1	1.15	1.15	1.05	1.3	0.975	1.05	0.925	1.325	1.225	0.95	1.05	1.4	1.2	1.1	1.225	1.325	1.3	0.8	0.95	1.25	1.1	1.15	1.4	1.25

*All the values are to be multiplied by 10^6 to get the actual data