NOTE TO USERS

uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Hegui Ye
AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.A.Sc. (Mechanical Engineering)
GRADE / DEGREE

Department of Mechanical Engineering
FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Solving the Combined Modular Product Scheduling and Production Cell Reconfiguration Problem: A Genetic Algorithm Approach with Parallel Chromosome Coding

TITRE DE LA THÈSE / TITLE OF THESIS

M. Liang
DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

B. Dhillon

X. Huang

Gary W. Slater
LE DOYEN DE LA FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES /
DEAN OF THE FACULTY OF GRADUATE AND POSTDOCORAL STUDIES

Solving the Combined Modular Product Scheduling and Production Cell Reconfiguration Problem: a Genetic Algorithm Approach with Parallel Chromosome Coding

A thesis submitted to
the School of Graduate Studies and Research
in partial fulfillment of the requirements of the degree of

Master of Applied Science

in

Mechanical Engineering

by

Hegui Ye

Ottawa-Carleton Institute of Mechanical and Aerospace Engineering
University of Ottawa
Ottawa, Ontario, Canada, K1N 6N5

Your file Votre référence
ISBN: 0-494-11465-7
Our file Notre référence
ISBN: 0-494-11465-7

Dedicated to

my wife, Huiqing Lin,

and

my son, Haomin Ye,

and

my daughter, Kaiyue Ye

for their unlimited patience

and understanding

Abstract

Manufacturers have to balance the desire for high productivity and the need for rapid responsiveness. These two interrelated problems can be solved by arranging the production of a family of modular products in Reconfigurable Manufacturing Systems (RMS). The potential benefits of RMS can be achieved by reconfiguring its structure in such a way that each configuration corresponds to one product variant in the same family. The successful implementation of this strategy lies in efficient scheduling of the system. However, little research has been done in addressing the scheduling issues in RMS. This study presents a modeling methodology to simultaneously determine the processing sequence of a family of modular products and select the optimal system configuration for producing each product variant. Due to the combinatorial nature of this problem, a genetic algorithm (GA) with parallel chromosome coding scheme is proposed to provide quick and near-optimal solutions to large problems. The efficiency of the proposed approach is demonstrated by comparing the computational results with that achieved using LINGO and by applying it to both small size and large size models.

Keywords: Modular products; Configuration Selection; Job sequencing; Genetic algorithm; Reconfigurable Manufacturing Systems

Acknowledgements

Many people have contributed towards the successful completion of this thesis. First and foremost I would like to express my deepest appreciation to my supervisor, Dr. Ming Liang. Without his constructive suggestions, overall guidance, thorough and patient examination of my writing, and financial support, I would never have completed this thesis work.

I would like to thank the members of my defense committee for spending their valuable time and effort on this thesis. I would also like to thank the academic staff for their support throughout my time in the university and my fellow students for their helpful ideas on my work

I am deeply indebted to my family members. My dear wife, Huiqing Lin, has made great sacrifices during my study period. It is her faithful support and spiritual encouragements that help me tide over this difficult period of time. I feel so sorry to my son, Haomin Ye, and my daughter, Kaiyue Ye, for the time I didn't play with them and for the duty that I didn't perform well as a father.

My genuine gratitude also goes to my parents, my brothers and sisters, my mother-in-law. Their love is an awesome force that drives me to the end of my study.

Table of Contents

List of Figures

List of Tables

Nomenclature

Indices

c	cell index, $c = 1,2,\cdots, C$ $(C=M)$
(c, j)	job index (the job of processing product variant j in cell c)
e, p	product variant position index, $e, p = 1,2,\cdots, N$
f	configuration index, $f \subset F_j^c$
i, j, k	product variant index, $i, j, k = 1,2,\cdots, N$
m	module index, $m = 1,2,\cdots, M$
l_m	module instance index, $l_m = 1,2,\cdots, L_m$
n	chromosome index
s	machine index

Parameters

CJF_{jk}^c	cost of reconfiguring production system from job (c, j) to job (c, k)
$CF_{f_j f_k}^c$	cost of changing cell configuration from f_j to f_k in cell c
CF	total system reconfiguration cost
CI	total machine idle cost
CI_{jf}^c	machine idle cost during processing job (c, j) if configuration f is used in cell c
CI_j^c	machine idle cost during processing job (c, j)
CM	total material handling cost
CM_j^c	material handling cost during processing job (c, j)
CW	total work-in-process holding cost
CW_j	work-in-process holding cost of product j

CT_j^c completion time of job (c, j)

D_j demand of product variant j

$DELT_j^c$ completion time difference between the last job and any other jobs for product j in cell c

F_j^c feasible set of configurations for processing job (c, j)

$g(n,q)$ gene q in chromosome n (used in serial chromosome coding only)

$g(n,q,1)$ gene q in chromosome n, the third subscript 1 represents job-position assignment

$g(n,q,2)$ gene q in chromosome n, the third subscript 2 represents job-configuration selection

J_{cj} job j in cell c

K an adjusting coefficient of determining the population size

M_{ml_m} instance l_m of module m

MT_s machine number s

MCT_j^c maximum completion time of all jobs for product j

N_P the population size

Q_j^c volume of job (c, j)

RM_s removal cost of machine number s

RA_s rearrangement cost of machine number s

$Pop(n,c)$ Sub-chromosome c in chromosome n

PT_{jf}^c processing time of job (c, j) using configuration f in cell c

PT_j^c actual processing time of job (c, j)

TC total number of manufacturing cells

TJ total number of jobs in a cell

TF_j^c the number of configurations included in the feasible set of configurations for job (c, j) in cell c

TN Total number of possible combinations of job sequences and cell configurations

TPC total production cost including the system reconfiguration cost, idle time cost of machine tools, material handling cost and WIP holding cost

UMH_{jf}^c distance that a component has to travel during processing one unit of job (c, j) if configuration f is used in cell c

UMI_{jf}^c number of idle machines during processing one unit of job (c, j) if configuration f is selected in cell c

UPT_{jf}^c number of time units required to process one unit of job (c, j) if configuration f is employed in cell c

u_h unit machine idle cost per time unit

u_m material handling cost per unit distance

u_w unit work-in-process holding cost

Variables

X_{jp}^c 1, if job (c, j) is in sequence position p in cell c; otherwise, 0

Y_{jpk}^c 1, if job(c, j) is in sequence position p and immediately followed by job (c, k) $(k \neq j)$; otherwise, 0

Z_{jf}^c 1, if configuration f is selected for processing job (c, j); otherwise, 0

Chapter 1

Introduction

Nowadays the marketplace is characterized as the one with strong desire for wide product varieties, small quantity of demand and fierce global competition. Low unit cost and high product quality are no longer sufficient to be successful in the competition. Rapid response to market change is the key to success in today's market environment. Some new philosophies in product design and production system design, including modular product design and reconfigurable manufacturing, have emerged to help manufacturers to gain competitive edges in the last decade.

1.1 Product Modularization

Product modularization is a product design technique that aims at clustering various components in a product into a certain number of building block or modules. The purpose of decomposing a product into several modules is to establish a modular structure. In an ideal modular structure, modules are physically and functionally independent. Each module has often more than one instance, varying in size, capacity, and even functionality. Usually different instances within the same module are exchangeable and the replacement of one instance with another does not affect other modules in term of physical and functional requirements. The combinations of different module instances generate a large number of product variants, which forms a family of modular products and gives manufacturers the flexibility to adjust quickly in the ever-changing market.

Manufacturers benefits from modular product design in many aspects. First, product variety can be substantially expanded to cater for different market segments with relatively small number of module instances. Second, the economies of scale can be achieved if several product variants are produced at the same time. This is because different instances within the same module usually share similar manufacturing resources and hence can be processed in relatively large quantities. Modular product design also

helps streamline logistics, make upgrade and maintenance easier, reduce lead time, and facilitate the recycle and disposal of products, etc.

1.2 Reconfigurable Manufacturing Systems

The marketplace is undergoing a major shift from mass production to mass customization. Mass production means to produce a single product at high volume for a long period of time. It is typically achieved by Dedicated Manufacturing Systems (DMS). DMS achieve high productivity usually by using multiple spindles in fixed directions to simultaneously perform several cutting tasks. The production lines are determined at the design stage, hence lacking the flexibility in response to dynamic market changes such as varying product volume and special customers' requirements. On the other hand, mass customization aims at better serving customers with products that are closer to their individual needs. Flexible Manufacturing Systems (FMS) can fulfill this goal since they can produce a variety of products using the same system by frequently changing fixtures, cutters and processing programs. However, the setup cost is high, hence it falls short in efficiency. In addiction, FMS tend to be over-equipped and much of the heavily invested capacity cannot be fully utilized (Mehrabi *et al* 2002).

To take advantage of both DMS and FMS, Reconfigurable Manufacturing Systems (RMS) are introduced. RMS possess the DMS' feature of high productivity, e.g., by using multi-task processing equipment and yet enjoy the flexibility of FMS because of their modular structure. The modular structure allows rapid reconfiguration and hence quick reaction to changes in production volume and product variety.

Because of the high productivity and fast responsiveness of RMS, they are especially suitable for producing a family of modular products whereas product variants are created from different module combinations. Each product variant belonging to the same family is to be produced by an RMS under its corresponding configuration.

1.3 Outline of Work

An industrial environment is dynamic in nature. The volume of each product variant and total number of product variants to be produced during a production horizon may

vary to a large extent. Unexpected events, such as late order arrivals, order cancellations, changes in order quantities, and delays in material shipments may also cause the initial schedule infeasible. Hence rapid response to unpredictable events is the key to scheduling the production of modular products in RMS. In addition, there may exist several feasible configurations for each product variant. These feasible configurations lead to different processing rates to produce a product, and the system changeover cost from one configuration to another is different. Therefore, another important issue in RMS is to determine the optimal configuration for producing a product variant because different selection policies can result in significantly different profits. Considering the dynamic nature of the market and the overall production cost, an integrated approach is highly desirable to concurrently scheduling the production of modular products and selecting the configuration for processing each product variant. For this purpose, a cost model is developed to minimize the total manufacturing cost of producing a batch of product variants during a production horizon. The combinatorial nature of the model precludes the use of any existing optimization software to solve meaningful sized industrial problems. As such, a genetic algorithm with parallel chromosome coding scheme is designed to provide a near optimal solution in a reasonable time range.

1.4 Organization of This Study

A brief introduction to the combined problem of scheduling modular products and selecting production cell configurations is given in this chapter. Chapter 2 provides a detailed review on modular products and their schedule problems in Reconfigurable Manufacturing Systems. The limitations of previous research and the motivation of this study are also presented. Chapter 3 elaborates several elements of the problem, i.e., the cell configuration, feasibility matrix, system configuration cost, material handling cost, machine idle cost and work-in-process holding cost. The steps of formulating the cost model that incorporates the above cost elements are also included in this chapter. Chapter 5 presents a genetic algorithm (GA) with parallel chromosome coding scheme. Some important components of the proposed algorithm including chromosome coding, initial population generation, parent selection, crossover and mutation are discussed in detail. The flowchart and overall procedure are also provided. In Chapter 5, an industrial case is

selected to validate the mathematical model and to illustrate the application of the GA method. The computational results of the GA approach are compared with that obtained using commercial software package LINGO8.0. The efficiency of the GA method is further examined by applying it to other computer randomly generated problems. The convergence behavior of the algorithm and the effects of GA parameters are discussed. Chapter 6 lists the findings of this study and suggests some future research areas.

Appendix A summarizes the LINGO solution to the cost model. The detailed Visual Basic code of the GA program with parallel chromosome coding scheme is presented in Appendix B. Appendix C lists the GA program with serial chromosome coding scheme for comparison purpose.

Chapter 2

Literature Review

Manufacturers have to balance the desire for high productivity (achieved by mass production) and the need for high responsiveness (done via mass customization). Decades ago, the manufacturing industry was successful due to mass production. A manufacturer could produce a single model of product with high volume using Dedicated Manufacturing Systems (DMS). By doing so, the same manufacturing resources can be repeatedly used for a long period of time and the production costs are kept minimum. In the era of mass production, customers have to "accept" the products provided to them rather than have their voice heard in choosing the products that meet their individual needs. From the customers' view points, however, products are ideally designed based on their specific requirements while still at affordable prices. This calls for mass customization, a great challenge to the traditional manufacturing principle of economies of scale. Global competition has forced manufacturers to make a huge change in business practices and structure to adapt to the paradigm shift in product and production system design.

2.1 Review on Product Modularization

The modular design concept has been used for many years. For example, automation equipment suppliers, such as FESTO, SMC and Norgren, provide modular actuators and grippers for building work-handling systems. Many manufacturers produce modular machine tools mainly by reconfiguring their existing machine sub-systems (Hu 1993).

Since the early modularity theory proposed by Ulrich and Tung (1991), much work has been done in this area. Fujita *et al* (1998) categorized research regarding product modularity into three levels: architecture level, configuration level and instantiation level. The architecture level concerns the establishment of modular architecture; the configuration level aims at developing standard platforms for product variants; and the

instantiation level involves selecting module instances under a given architecture and configuration.

Several strategies for the establishment of product modular architecture have been developed. Ulrich and Tung (1991) classified three types of modular architectures: component-swapping modularity, component-sharing modularity and bus modularity according to different interactions within a product. Based on Ulrich and Tung's classification of product modularity, Kusiak and Huang (1996) proposed a methodology to determine product modules while considering the tradeoff between cost and performance using a fuzzy neural network approach. Though their work provides a fundamental understanding of product modularity, only product functional and physical requirements are considered when modularizing a product.

In the area of product platform and configuration, various methodologies and measuring standards were developed. Nayak *et al* (2002) provided a methodology to identify a common product platform for individual product. Huang *et al* (2003) used Attribute Graphs to represent a family of products and the problem of identifying product platform was transformed into an issue of identifying the largest isomorphic sub-graph with the highest similarity. The concept of Mechanical Bus was proposed by Gu and Slevinsky (2003) to facilitate product platform design, and some important features of Mechanical Bus such as bus functions, locking, release mechanisms, positioning were identified.

At the instantiation level, most work is focused on the selection of module instances for product variants. Fujita (1999) used a binary integer formulation to find the optimal instance combinations for modular products. He (2002) further developed another optimization method to select module instances based on the similarity of module attributes. A general model for the selection of instances for a family of products was proposed by Gonzalez and Otto (2000).

The concept of modularity was also utilized in the construction of entire production systems. Rogers (1990) proposed the Modular Production Systems (MPS) concept to overcome the limitations resulting from a lack of modular machine standards. Rogers and Bottaci (1997) later outlined the main modules used for building an MPS. The matrix representation of the modularity problem was presented by Huang and Kusiak (1998).

They interpreted different types of modularity and used a decomposition approach to determine modules for different products. Khoo and Situmdrang (2003) proposed an approach based on an immune algorithm for the design of a multi-product assembly system. Grignon and Fadel (2004) introduced a GA method for optimizing the configuration of assembly components.

All of the above research deals with the modular product formation problem. The production issues of modular products are rarely mentioned.

2.2 Review on Manufacturing Systems

A brief literature survey suggests that there are different views on classifying the evolution of manufacturing systems. For example, Mehrabi *et al* (2002) described three major epochs of manufacturing systems: Pre-CNC epoch (pre-1960s), CNC epoch (1960-1990) and knowledge epoch (post-1990). In term of production scale, manufacturing systems can be divided into three major categories: Dedicated Manufacturing systems (DMS), Flexible Manufacturing Systems (FMS) and Reconfigurable Manufacturing Systems (RMS). The following review will be based on the latter classification.

2.2.1 Dedicated Manufacturing systems (DMS)

Dedicated Manufacturing systems (DMS) are designed for production of a certain product type at high production rate for a long period of time. DMS are mainly developed for mass production. It achieves high productivity using fixed manufacturing elements, such as multi-spindle machines, special designed tools, and constant speed transfer conveyors. DMS are cost effective since they are based on fixed automation and can operate at their full capacity. However, the production lines lack flexibility to respond dynamic market changes.

2.2.2 Flexible Manufacturing Systems (FMS)

Flexible Manufacturing Systems (FMS) mainly consist of general-purpose computer numerically controlled (CNC) machines and other programmable automation devices. FMS can manufacture a variety of products by changing fixtures, cutting tools and

processing programs. The material handling systems are typically characterized as robotic pick-and-place systems and automatic guided vehicles (AGV). Although FMS have improved the flexibility in response to changing production requirements, there still exist some limitations such as high capital costs, lack of efficiency and under utilization of capacity (or using more flexibility than needed).

One important variant of FMS should be mentioned, i.e., the Cellular Manufacturing Systems (CMS). CMS are generally designed based on the similar features of a fixed set of product families with stable demand for long life cycle. Manufacturing resources are partitioned into cells, each of which dedicated to a family of products. The predetermined structure of CMS, as pointed out by Benjaafar (1995) and Flynn and Jacobs (1986), restricts their flexibility in response to demand fluctuations whether in type or volume. Moreover, the cost of redesigning CMS and layout changes is too high. As a result, classical CMS may be known as unconfigurable manufacturing systems.

To overcome the structural limitations of physical CMS, the concept of a virtual manufacturing cell has been proposed (Simpson *et al* 1982). Unlike the physically fixed cellular layout using the Group Technology (GT) principles, the virtual cell defines a group of machines only in the system control software. It extends the concept of the traditional GT cells by allowing the sharing of machines with other cells that produce different part families (Irani *et al* 1993). The physical position of machines in a cell is no longer important and actually can be dynamically reconfigured. Although the virtual cell concept has some advantages over classical CMS, the inherited shortcomings of its cellular layouts, from the system reconfigurability point of view, are still unavoidable (Abdi and Labib 2003). These can be summarized as follows:

a) Uneven and low machine utilizations because of duplication of the same type of machines in different cells.
b) Low flexibility for product variety and demand fluctuation.
c) High changeover cost for cell reconfiguration, e.g. machine relocations.

Moreover, most of the virtual cell formation models assume that inter-cell material flows for some parts are negligible, which is usually not true in practice. This further restricts their applications.

2.2.3 Reconfigurable Manufacturing Systems (RMS)

To fill the gap between dynamic market demands and manufacturing system capability, Reconfigurable Manufacturing Systems (RMS) are introduced. RMS are designed at the outset for rapid adjustment of production capacity and functionality, in response to new market circumstances, by basic change of its structure as well as its hardware and software components (Mehrabi *et al* 2000). In contrast, Zhao *et al* (2000) considered RMS as ones with different structure configurations, each of which corresponds to a family of similar products.

RMS must be designed and built around parts of a product family that are being manufactured and provided only the flexibility needed for those specific parts (Mehrabi *et al* 2000). In this sense, RMS have the DMS' characteristic of high production rate and possess module structures to be quickly and reliably reconfigured for new product introduction and production volume change, i.e., the production line scalability (Son *et al* 2001). Hence RMS enjoy the rapid responsiveness of FMS. Because of the high productivity and fast responsiveness of RMS, they are especially suitable for production of a family of modular products.

RMS usually consists of more than two Reconfigurable Manufacturing Cells(RMC). A RMC can only be reconfigured for producing different instances of its corresponding module. However, the machine tools in a RMC possess modular structure and can be easily relocated either within a RMC or between different RMCs.

RMS are cost effective and responsive to market change, hence are considered as the cornerstones of new manufacturing paradigm in 21st century (Mehrabi *et al* 2002).

2.3 Review on Production Scheduling Problems

Product scheduling determines the sequence of products to be produced in a manufacturing system so that some production objectives are met. Scheduling objectives can be classified into two broad categories (Shanker and Modi 1999): (i) job related objectives – tardiness, makespan, waiting time, flow time, work-in-process and costs; and (ii) resource related objectives – utilization, idle time, and costs. Usually, scheduling decisions are made with more than one objective. There are a wide range of publications on the scheduling problem related to makespan, resource utilization, due dates, etc., but

the majority of research was focused on job shop scheduling problems (JSP). Some researchers have laid a solid foundation in this area (e.g., Van Wassenhove and Gelders 1980, Deckro *et al* 1982, Ghiassi *et al* 1984, Norbis and Smith 1988). Recently, Shanker and Modi (1999) addressed a multiple product scheduling problem with the objective of makespan minimization in a flexible manufacturing system. A branch and bound heuristic was proposed as a solution methodology for the integer linear problem formation. Stafford and Tseng (2002) developed two general models for a family of M-machine, N-job flowshop sequencing problems. One significant feature of their models is that a special variable was derived to link the sequence-dependent setup times to jobs and sequence positions. The same concept is extended to facilitate calculating the multi-cell sequence-dependent configuration costs in this study.

Another research area in production scheduling is the flexible job-shop scheduling problem (FJSP). FJSP is an extension of the classical JSP and inherits all of the difficulties and complexities of its predecessor JSP. However, it is more complex than JSP because of the additional requirement to determine the assignment of operations to machines. There are two major steps in FJSP: assign each operation to a machine, and schedule these operations to meet certain predefined objective criteria. Bruker and Schlie (1991) were among the first to address the FJSP. They developed a polynomial algorithm to solve a two-job flexible job shop scheduling problem. In literature, two types of approaches have been used for solving FJSP with more than two jobs: hierarchical approach and integrated approach. The former treats the operation assignment and sequencing separately whereas the latter considers them simultaneously. Brandimarte (1993) applied the hierarchical approach to solve FJSP. He first assigned operations to machines using some existing dispatching rules and then determined the operation sequence using a tabu search algorithm. Tung *et al* (1999) used a similar method for scheduling the production in a flexible manufacturing system. Kacem *et al* (2002) proposed a genetic algorithm for multi-objective FJSP. Later, they (2002) further developed a hybrid genetics algorithm and fuzzy logic method to find the pareto-optimization solutions. On the other hand, some researchers prefer the integrated approach. Hurink *et al* (1994) proposed a tabu search method to consider the operation assignment and sequencing at the same time. Similarly, Dauzere-Peres and Paulli (1997)

also used tabu search procedure to define a neighborhood structure in such a way that the distinction between reassigning and re-sequencing an operation is negligible. This approach was improved by Mastrolilli and Gambrardella (2002) by presenting two neighborhood functions.

It is well known that job shop scheduling problems are NP-hard (Garey *et al* 1976, Gonzalez and Sahni 1978). That means optimal solutions cannot be obtained in polynomial time. Hence the heuristic and stochastic search methods draw much attention recently. Selvam and Balasubramanian (1985) used the grouping algorithm to determine the job operation sequence based on the criterion that the sum of material handling cost plus the machine idle time cost is minimized. Some new results on simulated annealing applied to the job shop scheduling problem were reported by Kolonko (1999). Xia and Wu (2005) developed an effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems by combining Simulated Annealing (SA) and Particle Swarm Optimization (PSO) method. Rajendrana and Ziegler (2005) developed two ant-colony algorithms for minimizing total flow-time in flow-shop scheduling problems.

2.4 Literature on the Combined Problems

To stay profitable, manufacturers should consider both customers needs and economies of scale. Modular product design and RMS have made it possible to manufacture products in mass customization while maintaining the efficiency of mass production. Nevertheless, the potential benefits can only be materialized by dealing with modular design and RMS issues concurrently. Correspondingly, the scheduling tools to support this new style of manufacturing are different from the traditional ones. Therefore, literature review on the combined problem is done based on two sub-problems: (i) the relationship between modular products and RMS, and (ii) scheduling the production of modular products in RMS. These two aspects are discussed as follows.

2.4.1 Modular Products and RMS

Optimizing modular product design for reconfigurable manufacturing is recently emerging as an important trend in new product development. Yigit *et al* (2002) developed

a systematic methodology for optimal selection of module instances for a modular product manufactured in an RMS environment. The proposed formation is based on finding a trade-off between the quality loss due to modularity and the system reconfiguration cost. Yigit and Allahveri (2003) further expanded the model to cover the problem of optimal selection of instances for a family of modular products in RMS. Different customer demand and the order priorities were also addressed in their work, but only the reconfiguration cost was included in their model. The uncertainty of product demand and other production cost elements such as machine idle cost and material handling cost, etc., were not taken into account. Lee (1997) analyzed the manufacturing cost elements and the relationship between component designs and system reconfiguration costs, but his work emphasized primarily on finding similar processing routes among different components for manufacturing cost reduction. Abdi and Labib (2004) contribute an approach of grouping products into families based on operational similarities to facilitate the design of a RMS.

The study on this topic has been extended to the modular product assembly problem. Giusti *et al* (1994) described the main structural characteristics and the performance of a reconfigurable assembly cell. He and Kusiak (1997) decomposed the design of assembly systems into two sub-systems based on the structure of modular products. The configuration problem of the assembly system is then formulated and solved by a tabu search based algorithm. Travaini *et al* (2002) formalized a representation model and a method for assembly line configuration. Three fundamental models are included in their study: the first describes assembly line structures, the second illustrates component assembly features and the third generates all the assembly sequences. Yu *et al* (2003) presented a knowledge-based timed color object-oriented Petri net modeling method for a Reconfigurable Assembly Systems (RAS). They claimed that the configuration of RAS allows flexibility not only in assembling a variety of products, but also in changing the system itself.

The above studies only address the relationship between modular products and RMS. The production problems are still unanswered.

2.4.2 Production Scheduling of Modular Products in RMS

Most of the previous studies are limited to the design problems of modular products and RMS. The operational issues in manufacturing modular products with RMS were rarely addressed in the accessible literature. Reported work includes the studies by He and Kusiak (1997) and Khoo and Situmdrang (2003). He and Kusiak (1997) defined the problem of assembly configuration for modular products and devised an analytical model. Their model addresses two dependent sub-problems: operation assignment and product scheduling. The two sub-problems respectively deal with allocating the operations to stations (assembly line configuration) and sequencing the products to be processed by an assembly line. Khoo and Situmdrang (2003) recently solved the assembly configuration problem for modular products using an immune algorithm approach.

Xu (2004) proposed a cost model and genetic algorithm for the integration of module instance selection and assembly line design. Three sub-problems including module instance selection, assembly line balancing and manufacturing resource planning are addressed simultaneously in their model. In this regard, the proposed methodology bridges the gap between product modularization and assembly line design. But his model can only handle single modular product problems.

Recently, an age-based genetic algorithm was developed to solve the manufacturing cell formation and production scheduling problems for virtual cellular manufacturing systems (Mak *et al* 2005). The objective of the model is to minimize the total component travel distance, and the sum of the tardiness of all products. The issue of scheduling a family of modular product in RMS is still open for investigation.

2.5 Limitations of Previous Work

Although there is a vast collection of work on the product modularization, most of the previous studies are confined to either the product family structure issue or the modular product formation problem. Scheduling modular products for reconfigurable manufacturing is rarely considered in related research.

When manufacturers design RMS, there often exist several feasible configurations for each family (Yang and Peters 1998). These feasible configurations have different processing speeds and different changeover costs for changing the configurations. Hence

an important issue in RMS is the selection of optimal configuration for a family because selection policy has a significant impact on profits. To date, no formal model has been developed to address this problem.

In spite of the previous efforts of researchers, the problems of scheduling modular product operations and selecting cell configurations in RMS have been mostly carried out separately, hereby causing inconsistent, infeasible and even conflicting decisions. To improve decision quality, an integrated approach is proposed to simultaneously solving the two problems.

2.6 Motivation of This Study

As mentioned above, manufacturers are facing the problems of accommodating customers' desire for widening product variety, enhancing productivity and cutting down production cost. Modular product design provides a technical basis for expanding product variety, as discussed in Section 2.1. RMS help manufacturers to achieve high productivity (Section 2.2.3). The overall production cost reduction involves many factors in RMS environment. Considering these factors in isolation often causes infeasible and conflicting decisions. Therefore the existence of an RMS and product modules alone does not guarantee profitability. For this reason, this study is directed towards: a) the development of a mathematical model such that the combined problem of scheduling modular products and selecting cell configurations can be addressed simultaneously to improve operation synergy; and b) the design of an efficient algorithm to solve the proposed model.

Chapter 3

Problem Statement and Model Development

The necessity of scheduling the production of modular products in RMS has been discussed in Chapter 2. In this chapter, some key points of the scheduling issues in RMS, namely, the relationship between modular products and Reconfigurable Manufacturing Cells (RMC), and the connection between jobs and cell configurations, are elaborated. Detailed analyses of production cost components, i.e., system reconfiguration cost, machine idle cost, material handling cost and work-in-process cost, are then presented. The cost model will be developed based on the analyses.

3.1 Modular Products and Manufacturing Cells

Modular products are formed by, physically and functionally, independent modules. Each module has often more than one instance.

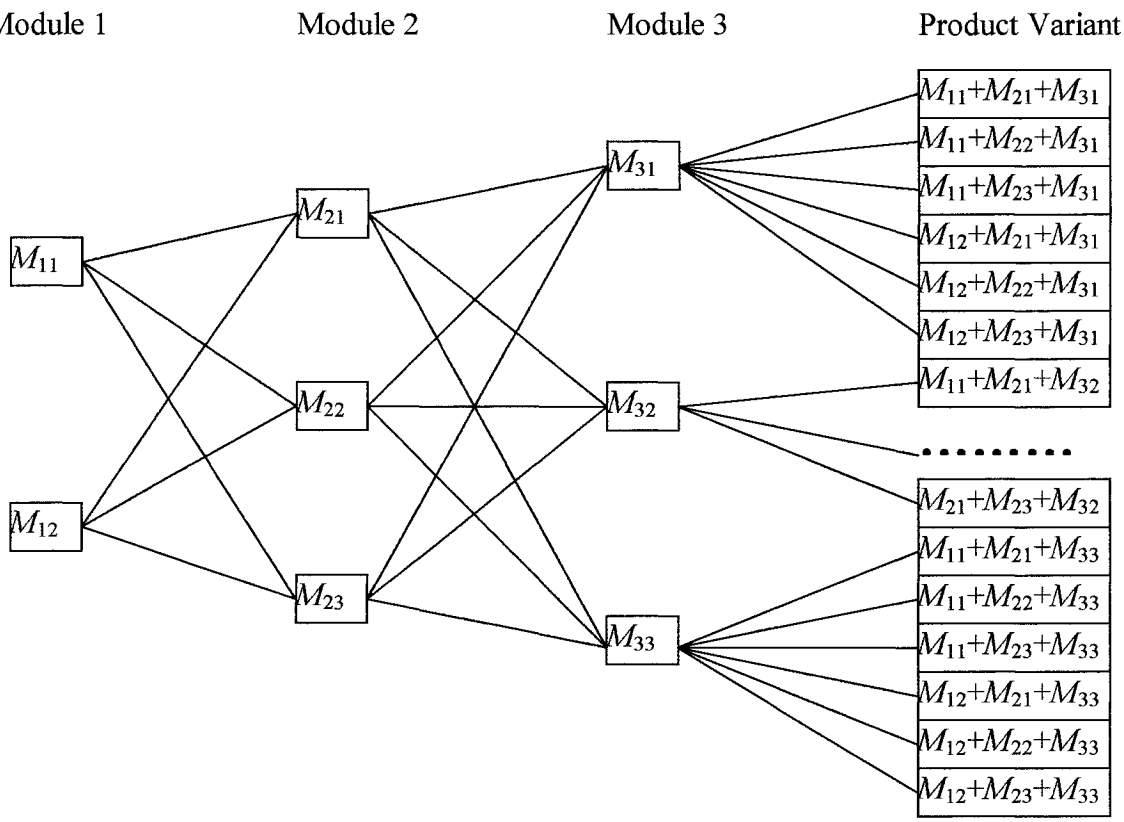


Figure 3.1 The structure of modular product

Figure 3.1 shows the structure of a modular product. Suppose a modular product consists of three modules M_1, M_2 and M_3. M_1 has two instances and each of M_2 and M_3 has three instances. As shown in Figure 3.1, the different combinations of these instances (i.e., $M_{1l_1} + M_{2l_2} + M_{3l_3}$; l_1=1, 2; l_2 =1, 2, 3; l_3 =1, 2, 3) will theoretically create 18 product variants (2 x 3 x 3 = 18), though not all of them are feasible. The relationship between modular products and manufacturing cells is clarified by using the following example.

Table 3.1 Components of modular products

Product variant	Components of product variant	M_1		M_2			M_3		
		M_{11}	M_{12}	M_{21}	M_{22}	M_{23}	M_{31}	M_{32}	M_{33}
1	$M_{12} + M_{23} + M_{31}$		Δ			Δ	Δ		
2	$M_{12} + M_{21} + M_{33}$		Δ	Δ					Δ
3	$M_{11} + M_{22} + M_{31}$	Δ			Δ		Δ		

Table 3.1 lists three possible product variants. Product variant 1 consists of M_{12}, M_{23} and M_{31}. Product variant 2 requires components M_{12}, M_{21} and M_{33}, and product variant 3 possesses M_{11}, M_{22} and M_{31}. Although these instances within the same module vary in sizes, capabilities and even functionalities, they usually share similar manufacturing resources and can be processed within the same manufacturing cell. Correspondingly, RMS designed for producing this family of products consist of three Reconfigurable Manufacturing Cells (RMC). The first RMC (R_1) can be quickly and cost effectively reconfigured to produce either M_{11} or M_{12}. The second RMC (R_2) can be restructured to process any instance of module M_1 (i.e., M_{21}, M_{22} and M_{23}). Similarly, the third RMC (R_3) can be used to produce instances of module M_3 (i.e., M_{31}, M_{32} and M_{33}). These three RMC are components of the whole RMS.

Table 3.2 Configurations of RMS for three product variants

Product Variant	Components of RMS	RMC							
		R_1		R_2			R_3		
		R_{1f_1}	R_{1f_2}	R_{2f_1}	R_{2f_2}	R_{2f_3}	R_{3f_1}	R_{3f_2}	R_{3f_3}
1	$R_{1f_2} + R_{2f_3} + R_{3f_1}$		Δ			Δ	Δ		
2	$R_{1f_2} + R_{2f_1} + R_{3f_3}$		Δ	Δ					Δ
3	$R_{1f_1} + R_{2f_2} + R_{3f_1}$	Δ			Δ		Δ		

The system configurations for producing product variants 1, 2 and 3 are shown in Table 3.2. It is assumed that each module is produced on a separate Reconfigurable Manufacturing Cell (RMC), which can be quickly reconfigured for producing different instances of the same module (Yigit and Allahverdi 2003). Consequently, each product variant is produced by its corresponding configuration of the RMS. For example, the first product variant $M_{12}+M_{23}+M_{31}$ is produced in RMS $R_{1f_2}+R_{2f_3}+R_{3f_1}$, where f_2, f_3 and f_1 are the cell configurations selected to process instances M_{12}, M_{23} and M_{31} respectively.

Suppose product variants 1, 2 and 3 are to be scheduled for production and each has a demand of D_j (j = 1, 2, 3). Since each instance is produced using its corresponding cell configuration, there are three jobs for each of these three manufacturing cells as shown in Table 3.3. In this study, it is assumed that each unit of product variant contains only one unit of instance from every module (Yigit *et al* 2002). Hence the volume of each job for a product variant equals the product demand. For example, product variant 1 consists of M_{12}, M_{23} and M_{31}. Therefore, the volume of each of J_{11}, J_{21} and J_{31} amounts to the demand of product variant 1 (D_1). As mentioned above, each module is to be produced in a separate RMC. For example, the first RMC can be reconfigured to produce any instance of module M_1 and the second can be reconfigured to process either M_{21}, M_{22} or M_{23}. Consequently, each module instance is produced by a separate configuration of RMS. This enables the manufacturers to be responsive to the dynamic demand. In Table 3.3, the instances contained in each job are listed in the second column under each cell.

Table 3.3 Jobs distributed in manufacturing cells

Product Variant	Product Demand	Cell 1		Cell 2		Cell 3	
		Job	Instance	Job	Instance	Job	Instance
1	D_1	J_{11}	M_{12}	J_{21}	M_{23}	J_{31}	M_{31}
2	D_2	J_{12}	M_{12}	J_{22}	M_{21}	J_{32}	M_{33}
3	D_3	J_{13}	M_{11}	J_{23}	M_{22}	J_{33}	M_{31}

It should be noted that the same instance may appear more than once in a single cell. Each appearance represents a different job since it comes from a different product variant. For example, instance M_{12} appears twice in cell 1, each belonging to a different

product variant and thus being treated as a distinct job in cell 1. Job 1 in cell 1 (denoted J_{11}) contains D_1 pieces of M_{12} and job 2 in cell 1 (J_{12}) has D_2 pieces of M_{12}. The reason of treating them as different jobs is that different completion times of job 1 and job 2 will be used to calculate the work-in-process (WIP) of product variants 1 and 2. Combining M_{12} in product variant 1 and M_{12} in product variant 2 as one batch of job in cell 1 with total quantity of $D_1 + D_2$ will make calculating WIP of product variants 1 and 2 impossible.

In this arrangement, each cell has three jobs. A total of nine jobs among three cells are to be scheduled. Scheduling the production of product variants 1, 2 and 3 can then be considered as scheduling the nine jobs in their corresponding cells. Please note that a job is represented by a (c, j) combination. For example, (1,3), (2,3) and (3,3) represent the three jobs of product variant 3 in three cells, i.e., J_{13}, J_{23}, and J_{33}. As will be discussed in Section 3.2, different schedules will lead to different system reconfiguration costs and WIP holding costs.

3.2 Relationship between Jobs and Cell Configurations

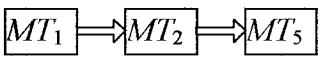
In the previous section, the characteristics of modular products and reconfigurable manufacturing systems are discussed. A family of modular products is decomposed into different modules. Instances within the same module usually share common manufacturing resources because of the similarity in processing operations. This means the majority of operations for each job in a cell are similar. The operation similarity between jobs allows easy reconfiguration of a production cell. The following sub-sections give the definitions of cell configuration, instance-configuration feasibility and job-configuration feasibility.

3.2.1 Cell Configuration

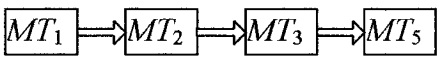
Cell configuration is defined as the machine layout in a cell that corresponds to the operation sequence of a job. In RMS, a manufacturing cell can be reconfigured to produce different instances of the same module.

Returning to the example in Table 3.3, there are three batches of jobs in cell 2: job 1 in cell 2 (J_{21}) with demand D_1 of M_{23}, job 2 in cell 2 (J_{22}) with demand D_2 of M_{21}, and job 3 in cell 2 (J_{23}) requiring D_3 pieces of M_{22}. In this case, three module instances (M_{21}, M_{22}

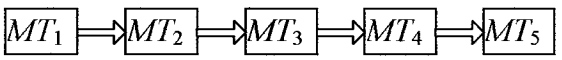
and M_{23}) of the same module M_2 are to be processed in cell 2. Each module instance requires certain operations to complete. Assume five operations needed to process M_{23} are carried out on machines 1 to 5 in the order as shown in Figure 3.1(c). Operation sequences for M_{12} and M_{22} are shown in Figures 3.1(a) and (b) respectively. Consequently, there are three configurations in cell 2, that is, configuration f_1 for processing M_{21}, configuration f_2 for producing M_{22}, and configuration f_3 for manufacturing M_{23}.



(a) Cell configuration f_1 for processing M_{21}



(b) Cell configuration f_2 for processing M_{22}



(c) Cell configuration f_3 for processing M_{23}

Figure 3.2 Cell configurations for three module instances in cell 2

If a configuration selected for processing a job has no idle-running machines, this configuration is considered as the most appropriate one for that job. Obviously, configuration f_1 is most appropriate for module instance M_{21} since all machines in configuration f_1 are fully engaged in producing M_{21}. Similarly, configurations f_2 and f_3 are the most appropriate configurations for processing module instance M_{22} and M_{23} respectively.

3.2.2 Instance-Configuration Feasibility

Instances within the same module are processed by different configurations of the same manufacturing cell. For each instance, there is a most appropriate configuration corresponding to it. However, this configuration may not be used to process other instances. As shown in Figure 3.1, only configuration f_3 in cell 2 can be used to produce module instance M_{23}. Hence the feasible set of configurations for instance M_{23} contains only one configuration f_3, i.e., $[f_3]$. Since configurations f_1, f_2 and f_3 are all capable of

processing M_{21}, the feasible set of configurations for M_{21} is $[f_1, f_2, f_3]$. Similarly, both f_2 and f_3 can be used to process M_{22}, so the feasible set of configurations for M_{22} is $[f_2, f_3]$.

Table 3.4 Instance-configuration feasibility matrix in cell 2

Cell 2	M_{21}	M_{22}	M_{23}
f_1	1*	0	0
f_2	1	1	0
f_3	1	1	1

*Note: value 1 means the configuration is capable of processing the associated module instance; 0 means the associated instance cannot be processed by the configuration.

The instance-configuration feasibility matrix for cell 2 is shown in Table 3.4. If the value of a configuration in its corresponding column of instance is 1, the configuration is capable of processing that instance. For example, f_3 =1 in all three columns of the instances, indicating f_3 is feasible for processing M_{21}, M_{22} and M_{23}.

3.2.3 Job-Configuration Feasibility

All configurations within the same cell that are capable of processing a job constitute the feasible set of configurations for that specific job. The feasible set of configurations for a job in a cell depends on which instance is included in the job.

Table 3.5 Job-configuration feasibility matrix in cell 2

Cell 2	J_{21}	J_{22}	J_{23}
f_1	0	1	0
f_2	0	1	1
f_3	1*	1	1

*Note: value 1 means the configuration is capable of processing the associated job; 0 means the associated job cannot be processed by the configuration.

Combining the information in Table 3.3 and Table 3.4, the job-configuration feasibility matrix in cell 2 can be derived as shown in Table 3.5. For example, job 1 in cell 2 (J_{21}) contains instance M_{23} and hence the feasible set of J_{21} is the same as that of M_{23}, i.e., $[f_3]$. The feasibility value of f_3 for J_{21} is 1. The configuration feasibility for other jobs can be obtained similarly.

The feasible set of configurations for different jobs in a cell could be the same. For example, the feasible sets of configurations for job 1 and job 2 in cell 1 are the same since they are to produce the same module instance M_{12}, i.e., $F_1^1 = F_2^1$. Here F_1^1 represents the feasible set of configurations for job 1 in cell 1 and F_2^1 is the feasible set of configurations for job 2 in cell 1. Both of them can be derived from the feasible set of configurations for module instance M_{12}.

3.3 Production Cost Elements

Although RMS can be reconfigured to produce different product variants within the same family, each system reconfiguration incurs machine relocation costs, retooling costs and other consequent adaptation costs. Therefore, it may not be a good idea to reconfigure a manufacturing system too often. Instead, it is desirable to prolong the life cycle of RMS as much as possible without compromising productivity and responsiveness.

The production cost consists of several components. When the production of one batch of instances is finished and the next batch is different from the previous one, system reconfiguration is required. It may involve the relocation of machine tools, retooling and reassignment of operators, etc. Hence each system changeover incurs a reconfiguration cost. A number of important cost components will be considered which include cell configuration cost (machine relocation cost), system reconfiguration cost, machine idle cost, material handling cost and WIP holding cost. These cost components are detailed in the following.

3.3.1 Cell Reconfiguration Cost

The cell configuration cost is specified as the cost of adding machines to and/or removing machines from a cell, i.e., the machine relocation cost (Lee 1997).

Consider cell 2 of the above example. The job sequence in cell 2 can start with any one of J_{21}, J_{22} and J_{23}. One possible job sequence is $J_{22} \rightarrow J_{23} \rightarrow J_{21}$. Cell reconfigurations from J_{22} to J_{23} and from J_{23} to J_{21}, if justified, will take place. Cell reconfiguration from one system to another is presented in Figure 3.1. For example, by relocating machine

MT_5 and adding MT_4, cell configuration f_2 is reconfigured to f_3. Similarly, cell configuration f_3 is reconfigured to f_1 by relocating machine MT_5 and removing machines MT_3 and MT_4. Note that relocating MT_5 involves two steps. The first step is to remove MT_5 from its original location and the second step is to rearrange it in a new location. Each step involves a cost. In order to differentiate the costs between these two steps, the removal cost and arrangement cost of machines are explained as follows:

(a) Removal cost of a machine means the cost of just removing the machine from its original location in the previous system reconfiguration.

(b) Arrangement cost of a machine is the cost of moving the machine to a new location in the same cell and setting it up in current system reconfiguration to be ready for production. Since this mostly involves machine relocation, arrangement cost is usually larger than removal cost of a machine.

During cell reconfiguration, one or both of the two costs may have to be considered for each machine. For example, both the removal cost and arrangement cost should be included for MT_5 when f_2 is reconfigured to f_3, but only arrangement cost needs to be considered for MT_4 during the same process.

MT_1	MT_2	MT_3	MT_4	MT_5
(RM_1, RA_1)	(RM_2, RA_2)	(RM_3, RA_3)	(RM_4, RA_4)	(RM_5, RA_5)
(4, 25)	(8,16)	(7, 19)	(5, 29)	(6, 17)

Figure 3.3 Removal and arrangement costs of machines

The removal and arrangement costs of machines 1 to 5 are shown in Figure 3.2. The first and the second values in the bracket are the removal cost and arrangement cost of that machine, respectively. To facilitate the calculation of these costs, vectors of removal cost and arrangement cost of machines are presented next for this example.

(a) Vector of removal costs of machines 1 to 5 = [4, 8, 7, 5, 6]

(b) Vector of arrangement costs of machines 1 to 5 = [25, 16, 19, 29, 17]

The cell reconfiguration cost is the sum of all the related removal and arrangement costs in the cell. When f_3 is reconfigured to f_1, machines MT_3 and MT_4 will be removed from this cell. The cost of removing MT_3 and MT_4 is RM_3+RM_4 =7+5=12. At the same time, MT_5 is relocated from its original location (position 5) to a new location (position 3

in the cell layout). The removal and arrangement cost of MT_5 is $RM_5+RA_5=6+17=23$. The total cost of reconfiguring f_3 to f_1 is therefore

$$CF_{f_3 f_1} = RM_3 + RM_4 + RM_5 + RA_5 = 7+5+6+17=35$$

If f_1 is reconfigured to f_3, MT_3 and MT_4 will be added to positions 3 and 4 and MT_5 is moved from position 3 to and set up in position 5. Hence the total cell reconfiguration cost from f_1 to f_3 will be

$$CF_{f_1 f_3} = RA_3 + RA_4 + RM_5 + RA_5 = 19+29+6+17=71$$

Similarly, the other configuration costs can be calculated as follows and the results are summarized in Table 3.6.

$$CF_{f_1 f_2} = RA_3 + RM_5 + RA_5 = 19+6+17 = 42$$
$$CF_{f_2 f_1} = RM_3 + RM_5 + RA_5 = 7+6+17 = 30$$
$$CF_{f_2 f_3} = RA_4 + RM_5 + RA_5 = 29+6+17 = 52$$
$$CF_{f_3 f_2} = RM_4 + RM_5 + RA_5 = 5+6+17 = 28$$

Table 3.6 Reconfiguration cost matrix in cell 2

$CF^2_{f_j f_k}$		To		
		f_1	f_2	f_3
From	f_1	0	42	71
	f_2	30	0	52
	f_3	35	28	0

3.3.2 System Reconfiguration Cost

The system reconfiguration cost is defined as the cost of switching production system from one product variant to another. As discussed in Section 3.1, a production system consists of several manufacturing cells. Hence the system reconfiguration cost depends on which cell configurations are to be selected for the jobs involved. It is the sum of all cell reconfiguration costs within a manufacturing system.

Figure 3.3 shows two consecutive jobs in a cell. Let F_j^c and F_k^c be the feasible sets of configurations for jobs (c, j) and (c, k), and f_j and f_k be configurations selected for

processing job (c, j) and job (c, k) respectively, i.e., $f_j \subset F_j^c$ and $f_k \subset F_k^c$. Different combinations of f_j and f_k will result in different cell reconfiguration costs.

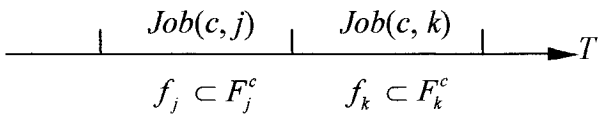


Figure 3.4 Configurations for job (c, j) and (c, k)

The cost of changing production system from job (c, j) to job (c, k) is the sum of all reconfiguration costs resulted from the different combinations of f_j and f_k. As mentioned before, f_j and f_k should be selected from their respective feasible sets. The reconfiguration cost from job (c, j) to job (c, k) is given by

$$CJF_{jk}^c = \sum_{f_j \subset F_j^c} \sum_{f_k \subset F_k^c} Z_{jf_j}^c \cdot Z_{kf_k}^c \cdot CF_{f_j f_k}^c \tag{1}$$

where $CF_{f_j f_k}^c$ is the reconfiguration cost from f_j to f_k in cell c, $Z_{jf_j}^c$ and $Z_{kf_k}^c$ binary variables. $Z_{jf_j}^c$ = 1, if configuration f_j is selected for processing job (c, j); otherwise, 0. $Z_{kf_k}^c$ = 1, if configuration f_k is selected for processing job (c, k); otherwise, 0. The total reconfiguration cost in cell c depends on the sequence of the jobs and the configurations selected for any two jobs that are arranged one after another, i.e., job (c, j) is immediately followed by job (c, k) in the job sequence as shown in Figure 3.4.

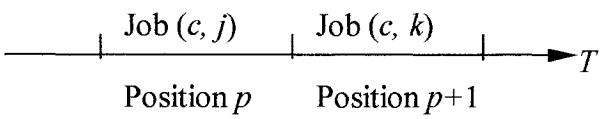


Figure 3.5 Job positions in the job sequence of cell c

The reconfiguration cost due to job changes in cell c can be expressed as

$$CF^c = \sum_{p=1}^{N-1} \sum_{j=1}^{N} \sum_{k \neq j}^{N} Y_{jpk}^c \cdot CJF_{jk}^c \tag{2}$$

where $Y^c_{jpk} = 1$ if job (c, j) is immediately followed by job (c, k), otherwise, 0. Variable Y_{jpk} is first used by Stafford and Tseng (2002) to calculate the job sequence-dependent setup time. It is extended here to deal with the multi-cell configuration problem by adding superscript c. The total configuration cost in all cells is given by

$$CF = \sum_{c=1}^{C} CF^c \tag{3}$$

3.3.3 Machine Idle Cost

Machine idle time is defined as the amount of time that a machine is not processing any element of a product. If a configuration selected for processing a job has no idle machines, this configuration is considered as the most appropriate one for that job. Obviously, in the above illustrative example, if the most appropriate configuration f_1 is selected to process module instance M_{21}, no idle time cost of machines incurs since all machines in configuration f_1 are fully engaged in producing M_{21}.Using any other configurations, for example f_3, from the feasible set of configurations $[f_1, f_2, f_3]$ to process M_{21} will result in machine idle-running. By comparing Figures 3.1(a) and (c), it can be seen that MT_3 and MT_4 are not needed in producing M_{21}. Since they are occupied here and cannot be used in other cells, a penalty cost of one machine unit will be imposed on machine MT_3 and MT_4 for processing each unit of M_{21}, causing a penalty cost of two machine units. If configuration f_2 as shown in Figure 3.1(b) is used to process M_{21}, only MT_3 is idle-running during the production of the whole batch of M_{21}. Therefore processing the same module instance by different configurations would lead to different machine idle costs.

The machine idle cost is determined by the number of idle machines and their idle-running time. The number of idle machines depends on which configuration is used to produce a job and the idle-running time of machines relies on job volume. The idle cost of all machine types during the period of processing job (c, j) is computed using the following equation

$$CI^c_j = u_h \cdot Q^c_j \cdot \sum_{f \subset F^c_j} Z^c_{jf} \cdot UMI^c_{jf} \cdot UPT^c_{jf} \tag{4}$$

where u_h is unit machine idle cost per time unit (assume it is the same for different machine types) and job volume $Q_j^c = D_j$. Binary variable $Z_{jf}^c = 1$, if configuration f in cell c is used to process job (c, j); otherwise $Z_{jf}^c = 0$. UPT_{jf}^c is number of time units required to process one unit of job (c, j) using configuration f in cell c. UMI_{jf}^c is the number of idle machines during the period of processing one unit of job (c, j) if configuration f is selected in cell c.

The total idle cost of all machines during the entire planning horizon is given by

$$CI = \sum_{c}^{C} \sum_{j}^{N} \sum_{f \subset F_j^c} u_h \cdot Q_j^c \cdot Z_{jf}^c \cdot UMI_{jf}^c \cdot UPT_{jf}^c \tag{5}$$

3.3.4 Material Handling Cost

To simplify discussions, it is assumed that the distance between any two adjacent machines is the same, i.e., one unit distance (Su and Hsu 1998, Selvam and Balasubramanian 1985). If a workpiece skips n machines to reach its next machine, then its transferring distance will be given by $(n + 1)$. As can be seen in Figure 3.1, if configuration f_3 is used to produce M_{21}, a workpiece will be transferred following the sequence $MT_1 \rightarrow MT_2 \rightarrow MT_3 \rightarrow MT_4 \rightarrow MT_5$, which requires the workpiece to travel 4 units of distance. If the most appropriate configuration f_1 is used to process M_{21}, a workpiece will follow a shorter transfer route, i.e., $MT_1 \rightarrow MT_2 \rightarrow MT_5$, only 2 units of distance. Obviously, different configurations would lead to different workpiece travel distances and different material handling costs.

In addition, a longer travel distance implies longer unit processing time of workpieces. If a job includes a large number of workpieces, the job completion time will be extended to a significant level as a result of the additional distance that a workpiece has to travel.

The material handling cost during the period of processing job (c, j) using configuration f in cell c can be written as

$$CM_j^c = u_m \sum_{f \subset F_j^c} Z_{jf}^c \cdot UMH_{jf}^c \cdot Q_j^c \tag{6}$$

where u_m is the material handling cost per unit distance, and UMH_{jf}^{c} is the distance that a component has to travel during processing job (c, j) if configuration f is used in cell c.

Consequently, the total material handling cost is the sum of CM_j^c for all jobs in all cells

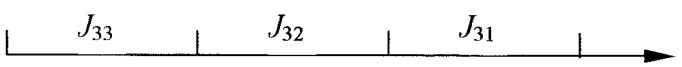
$$CM = \sum_{c=1}^{C} \sum_{j=1}^{N} \sum_{f \subset F_j^c} u_m \cdot Z_{jf}^{c} \cdot UMH_{jf}^{c} \cdot Q_j^c \tag{7}$$

3.3.5 Work-In-Process (WIP) Cost

One of the goals in manufacturing planning is to reduce the Work-In-Process (WIP) inventory cost as much as possible since inventory is viewed as a wasteful use of resources. To illustrate, consider the example in Table 3.1. Three product variants with different production volumes are to be processed in a RMS. Each product variant contains three batches of jobs (i.e., J_{i1}, J_{i2} and J_{i3}, and i=1,2,3). The processing sequences of these jobs for three product variants may or may not be the same. For example, product variant 1 consists of module instances M_{12}, M_{23} and M_{31} and the volumes of their corresponding jobs J_{11}, J_{21} and J_{31} are all equal to the demand of product variant 1 (D_1). These three jobs are distributed in different manufacturing cells and they may not be finished simultaneously. This will result in WIP holding cost.

(a) Job sequence in cell 1

(b) Job sequence in cell 2



(c) Job sequence in cell 3

Figure 3.6 Job sequences in three manufacturing cells

To find out the total WIP, it is necessary to analyze the job sequences in three cells simultaneously. One possible combination of job sequences in cell 1, 2 and 3 is shown in

Figure 3.5. The job sequences in cell 1, 2 and 3 are ($J_{11}\rightarrow J_{12}\rightarrow J_{13}$), ($J_{22}\rightarrow J_{21}\rightarrow J_{23}$) and ($J_{33}\rightarrow J_{32}\rightarrow J_{31}$) respectively. Product variant 1 consists of three jobs, J_{11}, J_{21} and J_{31} (referring to Table 3.3). J_{11} is scheduled as the first job in cell 1, J_{21} the second job in cell 2, and J_{31} the last in cell 3. The completion times of J_{11}, J_{21} and J_{31} differ from each other. If J_{31} is the last of the three to be completed, then the WIP of product variant 1 will be contributed by the holding time of J_{11} (the time difference between the completion times of J_{11} and J_{31}) and the holding time of J_{21} (the time difference between the completion times of J_{21} and J_{31}).

If J_{11}, J_{21} and J_{31} are scheduled as the first job in their respective cell, the completion time difference will be much smaller. It is obvious that changing job sequences in cells will result in different WIP holding cost. Suppose job (c, i) ($i \neq j$) is processed any time before job (c, j). Job (c, i) can be inserted in any job position before p, which is the position of job (c, j) in the sequence. Then the calculation of completion and waiting times can be carried out with reference to Figure 3.6.

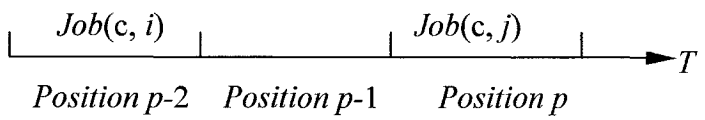


Figure 3.7 Job (c, i) precedes job (c, j) in the job sequence

Denoting the unit processing time of job (c, i) using configuration f in cell c as UPT_{if}^{c}, the processing time of job (c, i) can be expressed as

$$PT_i^c = Q_j^c \sum_{f \subset F_c^c} Z_{if}^c \cdot UPT_{if}^c \,, \ \forall c \tag{8}$$

The processing time of all jobs before job (c, j) in the sequence of cell c is given by

$$PT_{e<p}^c = \sum_{e=1}^{p-1} \sum_{i \neq j}^{N} X_{ie}^c \cdot PT_i^c \,, \ \forall c \tag{9}$$

where X_{ie}^c=1 if job (c, i) is assigned to position e (e<p) in the sequence in cell c; otherwise, 0.

The completion time of job (c, j) is the sum of the processing time of job (c, j) and all jobs that are processed before job (c, j) in the sequence

$$CT_j^c = PT_j^c + PT_{e<p}^c, \ \forall c, \forall j \tag{10}$$

The completion time of the last job for product variant j is the maximum value among all CT_j^c

$$MCT_j = \max\left(CT_j^c, \forall c\right), \ \forall j \tag{11}$$

The difference between the completion time of the last completed job and that of any other jobs for product variant j can be calculated by

$$DELT_j^c = MCT_j - CT_j^c, \ \forall c \tag{12}$$

Assuming the unit WIP holding cost u_w is the same for all jobs, the total WIP holding cost of product variant j is given by

$$CW_j = \sum_{c=1}^{C} DELT_j^c \cdot u_w, \ \forall j \tag{13}$$

The total WIP holding cost of all product variants is then determined by

$$CW = \sum_{j=1}^{N} CW_v = \sum_{j=1}^{N}\sum_{c=1}^{C} DELT_j^c \cdot u_w = \sum_{j=1}^{N}\sum_{c=1}^{C}\left(MCT_j - CT_j^c\right)\cdot u_w \tag{14}$$

Till now, the configuration dependent costs (i.e., the system reconfiguration cost, machine idle cost and material handling cost) and the job sequence dependent WIP holding cost are discussed separately. However they are related to each other. Figure 3.7 shows one possible job sequence and the feasible configurations for each job in cell 2.

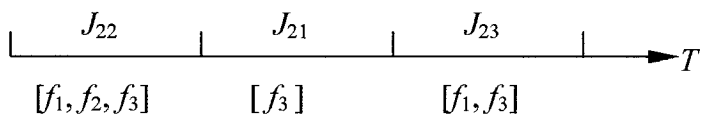


Figure 3.8 Job sequence and feasible set of configurations in cell 2

Note that configuration f_3 is feasible for J_{22}, J_{21} and J_{23}. If it is used for these three jobs in cell 2 during the whole planning horizon, the reconfiguration cost will be zero since no cell reconfiguration is needed. However, it should be pointed out that this arrangement will lead to higher machine idle cost and material handling cost, which have been discussed in Sections 3.3.2 and 3.3.3 respectively. On the other hand, the machine idle cost and material handling cost is minimized if the most appropriate configuration for each job is selected, i.e. f_2 for J_{22}, f_1 for J_{21} and f_3 for J_{23}. Nevertheless, this

arrangement requires two system configurations, i.e., $f_2 \rightarrow f_1$ and $f_1 \rightarrow f_3$, and results in a reconfiguration cost $CF_{f_2 f_1} + CF_{f_1 f_3}$. Therefore it is important to simultaneously consider the total reconfiguration cost, idle time cost of machine and material handling cost and to find the best compromise between these cost components.

3.4 Assumptions

To solve the integrated modular product scheduling and cell reconfiguration problem, the following assumptions are made:

a) Each product variant within the modular product family contains only one instance from each module.

b) All instances of a module are produced in one Reconfigurable Manufacturing Cell (RMC).

c) The unit material handling costs of all module instances are identical.

d) The machine relocation costs are independent of cells, i.e. they are the same in all cells.

e) The unit idle cost of each machine type is identical, regardless of its location in different cells.

f) The WIP holding cost per time unit for each instance is the same.

3.5 Mathematical Model

The objective is to minimize the total production cost, i.e., the sum of the system and cell reconfiguration costs, machine idle cost, material handling cost and WIP holding cost. The problem is accordingly formulated as

Min $TPC = CF + CW + CM + CI$ (15)

Subject to:

$$\sum_{j=1}^{N} X_{jp}^{c} = 1 \, (\forall c, \forall p) \tag{16}$$

$$\sum_{p=1}^{N} X_{jp}^{c} = 1 \, (\forall c, \forall j) \tag{17}$$

$$\sum_{f \subset F_j^c} Z_{jf}^c = 1 \, (\forall c, \forall j) \tag{18}$$

$$\sum_{f_j \subset F_j^c} \sum_{f_k \subset F_k^c} Z_{jf_j}^c Z_{jf_k}^c = 1 \quad (\forall c, \forall j, \forall k \neq j) \tag{19}$$

$$X_{jp}^c = \sum_{k=1, k \neq j}^{N} Y_{jpk}^c \; (p=1,2,\cdots, N\text{-}1; \;\; \forall c, \forall j) \tag{20}$$

$$X_{jp}^c = \sum_{k=1, k \neq j}^{N} Y_{k,p-1,j}^c \; (p=2,\cdots, N; \;\; \forall c, \forall j) \tag{21}$$

$$X_{jp}^c = 1 \text{ or } 0 \; (\forall c, \forall p, \forall j) \tag{22}$$

$$Y_{jpk}^c = 1 \text{ or } 0 \; (\forall c, \forall p, \forall j, \forall k \neq j) \tag{23}$$

$$Z_{jf}^c = 1 \text{ or } 0 \; (\forall c, \forall j, \; f \subset F_j^c) \tag{24}$$

The model may be explained as follows. Equations (16) and (17) ensure that each sequence position in a cell is filled by only one job, and that each job is assigned to only one sequence position in a cell. Equation (18) states that only one configuration is selected to process each job. Equation (19) guarantees that both configurations selected for two jobs that are arranged one after another are feasible.

Equations (20) and (21) link the sequence-dependent reconfiguration cost to jobs and sequence positions. They are the key points of this model. Equation (20) ensures that job (c, j) will be assigned to sequence position p if job (c, j) precedes the job in sequence position p+1. Equation (21) requires that job (c, j) be assigned to sequence position p if job (c, j) follows the job in sequence position p-1.

To simplify the explanation of the relationship between variables X_{jp}^c and Y_{jpk}^c, the cell index c is omitted for the time being. If four jobs are processed in a cell, a feasible job sequence, which is guaranteed by constraints (16) and (17), could be job 3 in position 1 (X_{31} = 1), job 2 in position 2 (X_{22} = 1), job 4 in position 3 (X_{43} = 1) and job 1 in position 4 (X_{14} = 1). All other X_{jp}'s will be zero. If all values of X_{jp}'s are given in Equations (20) and (21), there will be 40 constraints with value of either 1 or 0. These constraints will ensure that only $Y_{312} = Y_{224} = Y_{431} = 1$ and all other Y_{jpk}'s = 0. Obviously, Y_{312} = 1 means

that job 3 is the first job in the sequence ($X_{31} = 1$) followed by job 2, hence job 2 is in the second position ($X_{22} = 1$). $Y_{224} = 1$ implies that job 2 (in position 2) procedes job 4, therefore job 4 is in the third position ($X_{43} = 1$). $Y_{431} = 1$ denotes that job 4 (the third position) is followed by job 1. Job 1 can only be in the fourth position in the sequence ($X_{14} = 1$). Equations (20) and (21) guarantees that a feasible job sequence ($X_{31} = X_{22} = X_{43} = X_{14} = 1$) will derive a feasible set of job pairs with the correct job positions ($Y_{312} = Y_{224} = Y_{431} = 1$). The latter is critical for calculating the cell reconfiguration costs which have been discussed in Section 3.3.2. This concludes the model development.

Chapter 4

Solution Procedure

Due to its combinatorial nature, the model presented in Chapter 3 is unlikely to be solved in polynomial time. In fact, attempt has been made to solve small problems using LINGO 8.0 but the solution quality is questionable. It was also found that the computing time increases dramatically as the problem size grows. Hence, a genetic algorithm is developed to provide quick and near-optimum solutions to meaningful sized problems.

4.1 The Proposed Genetic Algorithm

Genetic Algorithm is based on the mechanism of natural population genetics and Darwinian survival of the fittest. The search process in optimization simulates the natural evolution of biological creatures and turns out to be an intelligent exploitation of a random search. The basic idea is to maintain a population of chromosomes with each chromosome encoding a candidate solution of the problem. Each candidate solution or chromosome is represented by an appropriate sequence of numbers. The quality or fitness of a chromosome is often simply the value of the objective function of the optimization problem. After evaluating the fitness of each chromosome, a new population of chromosomes is reproduced from the fittest chromosomes of the previous generation. This is usually accomplished by using crossover and mutation operations. By comparing the fitness values of the newly generated offspring with those in the previous population, the chromosomes with improved fitness will survive. The same procedure is repeated until the stopping condition is met. Eventually, a near-optimal solution can be found.

Since GA can operate without any underlying assumptions regarding continuity or differentiability of the underlying functions describing a product behavior, it has been widely used to solve many NP-hard optimization problems in a variety of areas and has proven to be very effective.

4.2 Chromosome Coding

One important issue in applying GA is the chromosome coding. It is the key factor affecting the search speed and efficiency of a genetic algorithm. Chromosome coding refers to the mapping of dependent variables in a mathematical formulation to the individual genes in chromosomes. Gene is the basic component in a chromosome. Its value and position in a chromosome represent essential information which will be passed on to offspring chromosome.

4.2.1 Serial Chromosome Coding

Conventionally, the serial chromosome coding scheme is often used. The genes in serial coding chromosomes are jointed one after another. Consider a system with three manufacturing cells and there are four jobs in each cell to be scheduled. Figure 4.1 shows a chromosome using the serial coding scheme.

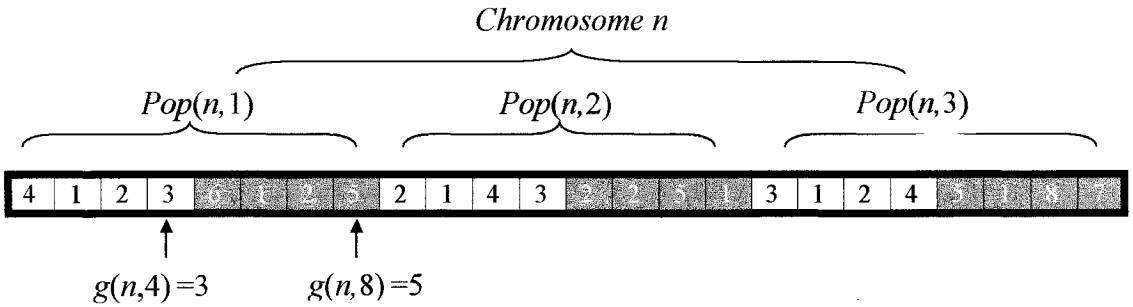


Figure 4.1 A chromosome using serial coding scheme

Based on the characteristics of the models, each chromosome *Pop*(n) (the nth chromosome in a population of size N_p, and N_p>2) is divided into three sub-chromosomes: *Pop*(n,1), *Pop*(n,2) and *Pop*(n,3). Sub-chromosome *Pop*(n,1) represents a solution for cell 1 while sub-chromosome *Pop*(n,2) denotes the same for cell 2, and so on. Each sub-chromosome is further divided into two portions: the first half portion represents the job sequence (the first four genes) and the second half denotes cell configurations selected for jobs (the following four genes). The relative gene position in each portion also stands for job number. For example, in sub-chromosome *Pop*(n,1), the value of the 4th gene in the first portion is 3, i.e., g(n,4)=3, which means that job 4 is the third job in cell 1. Hence the first four genes of *pop*(n,1) are interpreted as job 2 being placed in the first position followed by, in the same order, jobs 3, 4 and then 1.

The interpretation of genes in the second half portion of a sub-chromosome is not so straightforward. The selected configuration should be mapped to its corresponding job. For instance, the 8th gene (or the 4th gene in the second portion) of sub-chromosome $pop(n,1)$ is $g(n,8)=5$. It means configuration number 5 is selected for its corresponding job which is job 4 represented by the value of the 4th gene, i.e., $g(n,4)$, and has been placed in the third position. Combining these two pieces of information, it can be concluded that job 4 is the third job in cell 1 ($g(n,4)=3$) and configuration number 5 is selected for job 4 ($g(n,8)=5$). Similar interpretations can be made for the other genes.

4.2.2 Parallel Chromosome Coding

A symbolic parallel chromosome representation is used in this study due to the nature of the problem. Each chromosome consists of C sub-chromosomes, where C is the number of cells in the manufacturing system. For example, chromosome n as shown in Figure 4.2 has three sub-chromosomes. It represents a manufacturing system that has three manufacturing cells. Sub-chromosome $Pop(n,1)$ represents the job scheduling and configuration selection in cell 1 while sub-chromosome $Pop(n,2)$ denotes that information in cell 2, and so on.

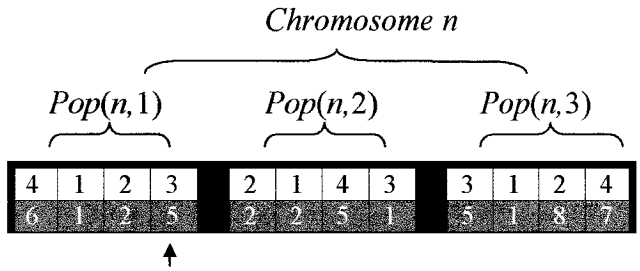


Figure 4.2 A chromosome using parallel coding scheme

Each chromosome contains two sets of genes. The first set of genes represents job-position assignment and the second one denotes job-configuration selection. Each gene contains four pieces of information. The first subscript n is the chromosome number. The second subscript is the gene position number in the chromosome and it also stands for job number. The third represents job-position assignment if its value is 1, or for job-

configuration selection if its value equals 2. The gene value itself denotes the job position in the sequence if the third subscript is 1, or the configuration selected for its corresponding job if the third subscript is equal to 2. For instance, $g(n,4,1)=3$ indicates that job 4 is the third job in cell 1, and $g(n,4,2)=5$ means that job 4 uses configuration number 5 in cell 1.

The job sequences and configurations selected for jobs in all three cells coded in Figure 4.2 is summarized in Table 4.1. In cell 1, for eaxmple, job numbers 1, 2, 3 and 4 are the gene position numbers. Job sequence 4→1→2→3 corresponds to the gene values in the first sub-gene set of $Pop(n,1)$. Configurations selected for jobs 1, 2, 3 and 4 are 6, 1, 2 and 5 respectively, which are the values of the four genes in the second sub-gene set of $Pop(n,1)$.

Table 4.1 Interpretation of chromosome n

Cell 1	Job number	1	2	3	4
	Job sequence	4	1	2	3
	Variable X^1_{jp}	$X^1_{14}=1$	$X^1_{21}=1$	$X^1_{32}=1$	$X^1_{43}=1$
	Configuration Number	6	1	2	5
	Variable Z^1_{jf}	Z^1_{16}	Z^1_{21}	Z^1_{32}	Z^1_{45}
Cell 2	Job number	1	2	3	4
	Job sequence	2	1	4	3
	Variable X^2_{jp}	$X^2_{12}=1$	$X^2_{21}=1$	$X^2_{34}=1$	$X^2_{43}=1$
	Configuration Number	2	2	5	1
	Variable Z^2_{jf}	Z^2_{12}	Z^2_{22}	Z^2_{35}	Z^2_{41}
Cell 3	Job number	1	2	3	4
	Job sequence	3	1	2	4
	Variable X^3_{jp}	$X^3_{13}=1$	$X^3_{21}=1$	$X^3_{32}=1$	$X^3_{44}=1$
	Configuration Number	5	1	8	7
	Variable Z^3_{jf}	Z^3_{15}	Z^3_{21}	Z^3_{38}	Z^3_{47}

4.3 The Main Modules of the GA Program

The implementation of the GA program involves the following steps: (1) generate the initial population; (2) select parent chromosomes for reproduction; (3) perform crossover operation on selected chromosomes; (4) conduct mutation operation; (5) decide when to

stop the program and output results; (6) incorporate the cost components in the objective function. These steps are detailed in the following sections.

4.3.1 Generate the Initial Population

The first phase in the algorithm is to generate the initial population. It includes all chromosomes that are generated at the very beginning for the subsequent operations in GA program. Although some heuristic methods and pre-selection techniques have been proposed to produce initial chromosomes, they are mainly restricted to specific research areas. Population initialization is usually done randomly. Kim *et al* (2003) has shown that a GA with randomly generated initial population outperforms its counterpart with pre-selected starting population. In this study, individual chromosomes are all randomly created. Obviously, the sequence of the data representing job position and configuration selected for a job is essential to find a feasible solution for the established model. Hence a randomly generated chromosome is subject to all constraints imposed on it. There exist two constraints for each individual chromosome.

(1) Each gene position can only be filled with one job and each job in a cell should be assigned to one gene position of the first or the upper sub-gene set (e.g., see Figure 4.2).

(2) The value of each gene in the second or lower sub-gene set should be taken from the job-configuration feasibility matrix of its corresponding cell.

The population size, N_p, refers to number of chromosomes to be produced initially and in each of the following generations. It is a problem-dependent parameter and, in this study, is specified as $N_p = K*(TC+TJ)$, where K is an adjusting coefficient, TC and TJ are the total number of cells and total number of jobs in each cell respectively. K is selected between 2, 4 and 6 based on the initial computational experience. The purpose of choosing different K's is to test the effect of population size on the performance of the GA.

The initialization process involves two sub-procedures: create genes representing job-position assignment, i.e., the first sub-gene set and then generate genes denote configuration selections, i.e., the second sub-gene set. The detailed process is as follows:

Step 1 *Generate the first chromosome*

Set n=1 and initialize related parameters.

Step 2 *Create sub-genes for job-position assignment*

Step 2.1 Generate the first sub-chromosome for cell 1. Set c=1 and initialize related parameters.

Step 2.1.1 The value of the first gene of sub-chromosome *Pop*(n, c) in the first sub-gene set can be any job number in its corresponding cell. Put the value in a final data array and activate its counter. At this point, the counter reads "1" since there is only one data entry in the array.

Step 2.1.2 Randomly generate a feasible job number for the next gene of *Pop*(n, c) in the first sub-gene set.

Step 2.1.3 Check this job number with those in the final data array one by one. If the job number exists in the final data array, discard it and go back to Step 2.1.2. If not, put the newly generated job number into a temporary data array and increase its counter value by one.

Step 2.1.4 Repeat Step 2.1.3 and check the value of the temporary array counter. If it is equal to that of the final array counter, the newly created job number has been checked with every data in the final array. This guarantees the newly created job number is different from everyone in the final data array.

Step 2.1.5 Place the newly created job number in the final data array. Clear the temporary data array for next gene.

Step 2.1.6 Repeat Steps 2.1.2 to 2.1.5 until all jobs in cell c are assigned.

Step 2.2 Set c=c+1 and repeat Steps 2.1.1 to 2.1.6 until all jobs in all cells are assigned.

Step 3 *Create sub-genes for job-configuration selection.*

Step 3.1 Generate the first sub-chromosome for cell 1. Set c=1 and initialize related parameters.

Step 3.2 Randomly generate a feasible configuration number for each gene in the second sub-gene set of sub-chromosome *Pop*(n, c).

Step 3.2 Set c=c+1, and repeat Step 3.2 until every job in each cell has a feasible configuration.

Step 4 *Generate next chromosome*

Let n=n+1, and repeat Steps 2 to 3 for the next chromosome.

Step 5 Stop the process when the maximum allowable number of chromosomes have been generated and calculate the fitness values for these initial chromosomes.

At this point, all chromosomes in the initial population pool satisfy the feasibility constraints.

4.3.2 Select Parent Chromosomes for Reproduction

The second phase in the GA procedure is to choose parent chromosomes for reproduction. Different techniques such as random selection, roulette wheel selection, tournament selection and stochastic universal sampling (SUS) have been used to select parent chromosomes. The initial test of the GA program shows that parent selection has no significant effects on the final results. Hence, only the first two methods are used to test the performance of the proposed GA in this study.

Random selection, as suggested by its name, is to choose parents without considering their fitness values. The roulette wheel selection is a little more complex and needs some explanations.

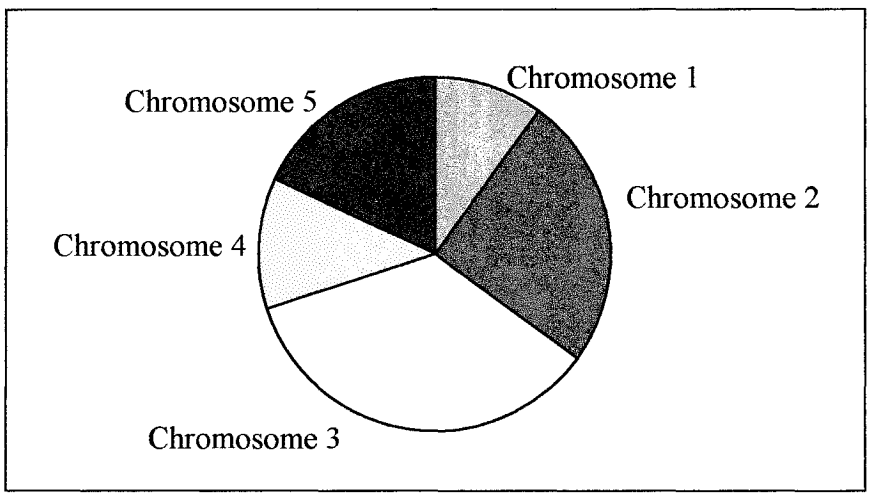


Figure 4.3 Roulette wheel selection

Imagine that all chromosomes in the population are spaced across the circumference of a roulette wheel with the size of an individual chromosome's slot proportional to its fitness value. The smaller the value is, the larger the slot it occupies in the roulette wheel. This will ensure that chromosome with smaller fitness value be given higher priority to be selected since the objective is to minimize the total production cost. When the wheel is

spun, chromosome at the position where the wheel stops is selected. Apparently, the probability of the wheel stopping in any particular slot is proportional to the slot size, i.e., the inverse of the fitness value of the corresponding individual chromosome. For instance, consider a population with 5 chromosomes as shown in Figure 4.3. Chromosome 3 is the most likely one to be chosen since it occupies the largest portion of the roulette wheel.

The roulette wheel selection involves the following steps:

Step 1 Calculate the fitness of each chromosome and sum them to obtain the total fitness of all the chromosomes in the population.

Step 2 Normalize the fitness of each chromosome and calculate its fitness proportion in the whole population.

Step 3 Sort all chromosomes based on their fitness proportions, the smaller the fitness proportion of a chromosome, the closer it is arranged to the start point in the queue.

Step 4 Randomly generate a number in the interval (0,1).

Step 5 Select the chromosome whose proportion plus the total proportions of its preceding chromosomes is greater than or equal to the random number generated in Step 4.

Step 6 Repeat steps 3 and 4 until the number of chromosomes required for reproduction is reached.

Roulette wheel selection chooses population members in a manner proportional to the "goodness" of their fitness values.

4.3.3 Perform Crossover on Selected Chromosomes

Crossover is the process of combining the genes of one chromosome with those of another to create offspring that inherit traits of both parents. Note that in this problem, the number of jobs in a cell to be sequenced is fixed. Consequently, chromosomes in the population are of the same length.

Crossover operation involves two major steps: determine the number of chromosomes for crossover and decide which crossover procedure to use. Determining the number of chromosomes for crossover is a critical issue in developing an efficient genetic algorithm.

Choosing a higher percentage of chromosomes for crossover allows a deeper exploration of the solution space; but if the percentage is too high, there is a possibility of wasting time in exploring bad regions of the solution space. In this study, all child chromosomes generated by crossover operation are put in a temporary population pool until the number of child chromosomes reaches the initial population size.

In literature many crossover procedures have been investigated. The simplest way to perform the crossover operation is to use one-point or two-points cutting policy. For one-point cutting, offspring are generated by copying the genes before the cutting point from one parent and by duplicating the genes after the cutting points from the other parent. The two-points crossover is done by maintaining the genes before the first cutting point and after the second cutting point and by swapping the genes between the two cutting points.

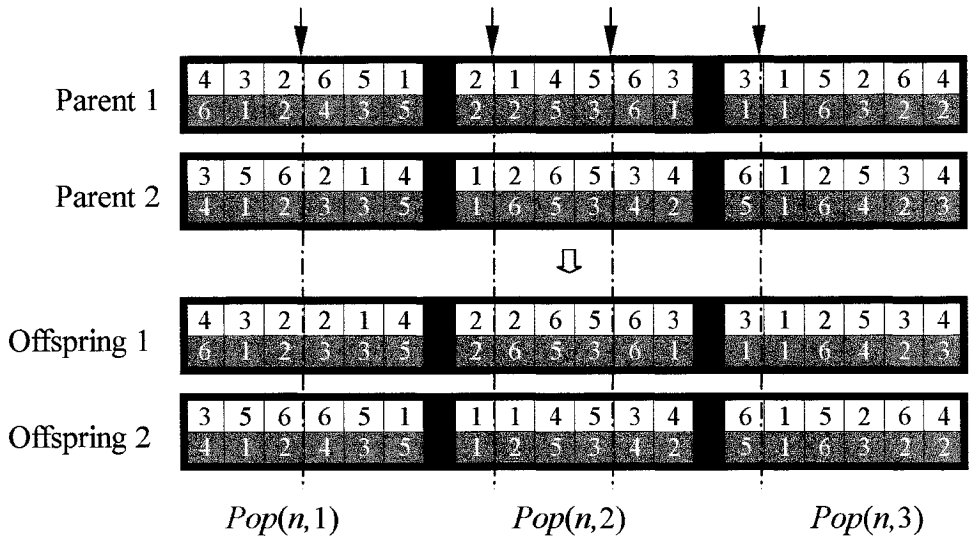


Figure 4.4a Standard crossover operation

Applying these standard crossover operations on current chromosomes does not necessarily create feasible offspring in this case. For example, consider the problem of scheduling three cells and each cell having six jobs. Two chromosomes (parent 1 and parent 2) are randomly generated as shown in Figure 4.4a. Assume the standard one-point crossover is applied on sub-chromosome $Pop(n,1)$ and the cutting point is the third gene. For offspring 1, the genes before the cutting point are the same as those in parent 1 and the genes after the cutting point are obtained from those in parent 2. The job sequence denoted by offspring 1 becomes 4-3-2-2-1-4. This sequence means jobs 1 and 6 (denoted

by the gene position, i.e., the 1st and 6th genes) are assigned to the same position 4, and jobs 3 and 4 are loaded as the second job in the sequence. This arrangement obviously violates the constraint that each job can only be assigned to one sequence position. Standard two-point crossover is applied for *Pop*(*n*,2) in Figure 4.4a. Both offspring contain duplicated genes in the first sub-gene set that implies they are infeasible. Hence, standard crossover operation does not guarantee that offspring are feasible.

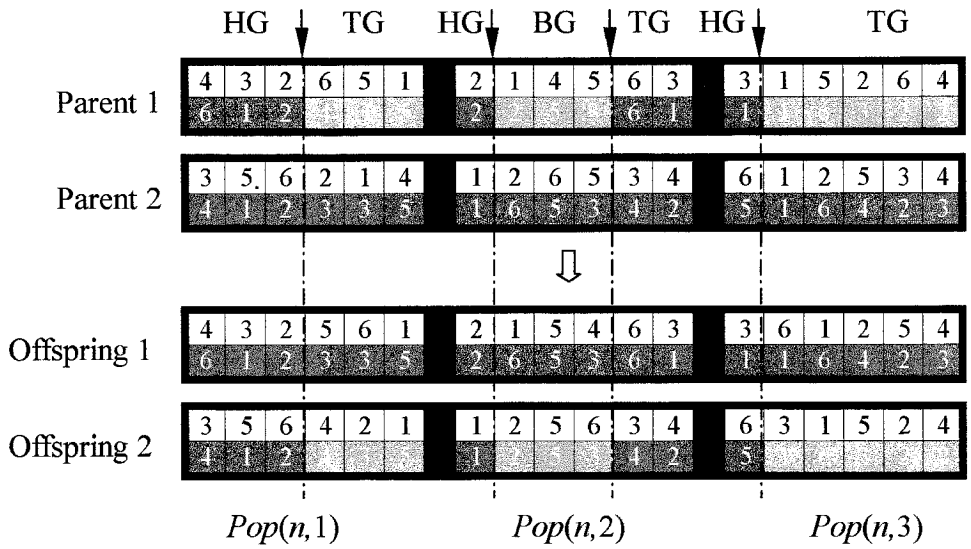


Figure 4.4b Re-ordered crossover operation

To produce feasible offspring out of two parents in such a way that the children inherit as much as possible of the meaningful information from both parents, a re-ordered crossover (Yeo *et al* 1996) is applied in this study. To facilitate the discussion of this crossover procedure, each sub-chromosome is divided into different segments as shown in Figure 4.4b and explained in the following:

(1) If one-point cutting policy is applied in a sub-chromosome, it is split into two segments: genes before the cutting point, as indicated by a solid downwards arrow, are called the Head Genes (HG) and those after the cutting point are defined as the Tail Genes (TG). For example, each of sub-chromosomes *Pop*(*n*,1) and *Pop*(*n*,3) has two segments.

(2) If two-point cutting policy is used, each sub-chromosome is divided into three segments: genes before the first cutting point and after the second cutting point are

the head and tail genes respectively. Genes between the two cutting points are the Body Genes (BG). For instance, sub-chromosome *Pop*(*n*,2) has three segments: the Head Genes (HG), Body Genes (BG), and Tail Genes (TG).

Note that both one-point and two-point cutting policy can be applied to the same chromosome. The re-ordered crossover operation involves two steps:

(a) Create the first sub-gene (or upper sub-gene) set, i.e., job sequencing set for each sub-chromosome using the re-ordered crossover operation.

One-point cutting policy is applied in *Pop*(*n*,1). The head genes from both parents are copied to their children (offspring 1 and offspring 2) directly. The tail genes in the first sub-gene set of offspring 1 are obtained from the tail genes on parent 1 (i.e., 6, 5 and1), but they have been re-ordered according to the sequence in which they appear on parent 2, which is 5 ahead 6, and 6 before 1, i.e., 5→6→1. In the same manner, the tail genes in the first sub-gene set of offspring 2 are copied from those of parent 2 (i.e., 2, 1 and 4), but they are sorted according to the order in which they show on parent 1, i.e., 4→2→1. Both offspring are constructed in a way that guarantees feasibility while still inherits the job-position information from their parents.

Two-point cutting method is used in *Pop*(*n*,2). The head and tail genes are duplicated from their parents, but the body genes are formed using the re-ordered crossover operation.

For sub-chromosome *Pop*(*n*,3), one-point cutting crossover is performed.

(b) Create the second sub-gene (or bottom sub-gene) set, i.e., job configuration set for each sub-chromosome using the standard crossover operation.

In this procedure, the head genes are transferred to their children without any modifications. Exchanging the tail genes in the second sub-gene set from both parents does not violate the feasibility constraints. For example, the fourth genes in parents 1 and 2 are $g(1,4,2)$ =4 and $g(2,4,2)$ =3 respectively. Combining these two pieces of information, it is known that job 4 can use configurations f_4 and f_3. Performing standard crossover operation on parents 1 and 2, the fourth genes in offspring 1 and 2 becomes $g(1,4,2)$ =3 and $g(2,4,2)$ =4 respectively. It still convey the same meaning that both configurations f_4 and f_3 are feasible for job 4. Similar analysis can be conducted on other tail genes in the second sub-gene set. Switching the tail genes between two parents just

changes the configurations selected for their corresponding jobs, but they are still feasible.

Preserving feasibility drastically reduces computational effort and makes the next decision, replacing parents in the population with children, much less cumbersome.

4.3.4 Perform Mutation Operation

Mutation is another important genetic operation that alters one or more gene values in a chromosome from its initial state. With these new gene values, genetic algorithms may be able to come up with a better solution than was previously possible (Leu *et* al 1994). Therefore, mutation operation aims at preventing all solutions in the population from being trapped in a local optimum during the optimization process.

Different mutation rates are often employed to test their effects on the performance of a genetic algorithm. A mutation rate has to be carefully defined for each problem since it controls the pace at which genes' values are changed. If the rate is too low, most of the offspring remains unchanged; but if the rate is too high, the offspring will start losing their resemblance to the parents. The mutation rate is usually below 0.15 (Kamrani and Gonzalez 2003). In this study, four mutation rates, i.e., 0.01, 0.02, 0.05 and 0.1 are used to test their behavior in the GA program. In addition, mutation rate is an important GA parameter that is used to determine the number of chromosomes to mutate (MutNum).

MutNum = Int(MutRate * (PopSize * (*TC* + *TJ*) * Rnd())) + 1

where MutRate is the mutation rate, PopSize is the population size, *TC* and *TJ* are respectively the total number of manufacturing cells and the total number of jobs in a cell, Int() is used to round off the value to the nearest integer number and Rnd() is to randomly generate a value between 0 and 1. At least one chromosome is selected for mutation in each iteration.

To ensure all constraints are satisfied, a particular variant of mutation called "swapping mutation" (Pirlot 1996) is used.

The swapping procedure is conducted on the first sub-gene set i.e., the job sequencing set. With this method, a mutation point is first decided randomly, e.g., the 4^{th} gene as shown in Figure 4.5(a). Next a feasible value representing the job position is randomly generated for the gene at the mutation point, assuming the gene value is 3. The temporary

chromosome is shown in Figure 4.5(b). Note that this chromosome becomes infeasible because the same value for genes 2 and 4 implies that jobs 2 and 4 are assigned to the same job sequence position 3. Therefore, the third step in the swapping mutation operation is to make this chromosome feasible. This is done by searching the genes before and after the mutation point to find the one whose value matches the newly generated one. In this case, it is the second gene $g(n,2,1)=3$. Now insert the original value of the gene at the mutation point, that is 6, in the second gene position and the chromosome becomes feasible.

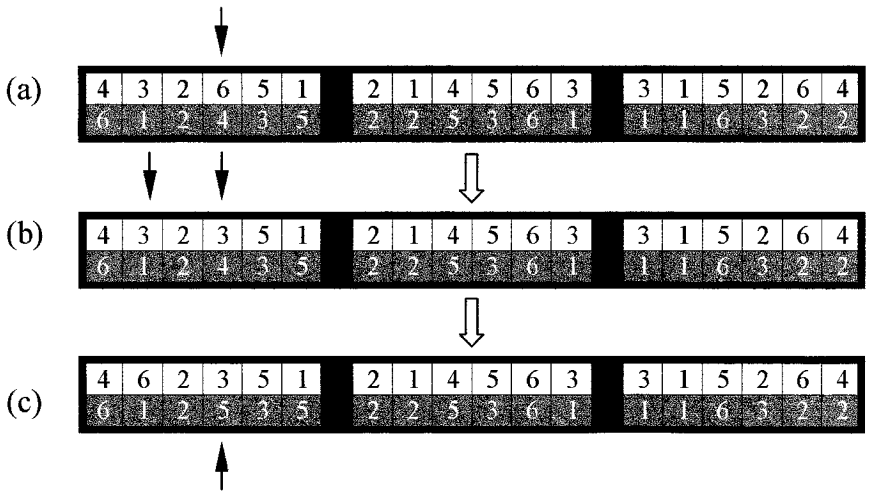


Figure 4.5 Mutation operation

The mutation procedure for the second sub-gene (i.e., the job configuration sub-gene) set is to randomly generate a feasible configuration number for the gene at the mutation point. For example, if the feasible set of configurations for job 4 is (2,4,5), then the value of gene 4 could be any of the three configuration numbers, i.e., 2, 4 and 5. In this example, assume it is 5. Then the original number "4" in the second sub-gene set is replaced by "5". This finished the whole swapping mutation procedure. The final chromosome is shown in Figure 4.5(c).

The fitness value of the new child is compared with that of its parent. If improved, the parent chromosome is replaced by its child. Otherwise, the child chromosome is discarded. The mutation procedure is repeated until the number of chromosomes to mutate (MutNum) is reached.

4.3.5 Incorporate the Cost Elements in the Objective Function

There are four cost elements in the objective function: the machine idle cost, material handling cost, system reconfiguration cost, and WIP holding cost. These cost elements have to be incorporated in the objective function.

The inclusion of the first two cost elements in the GA program is quite straightforward. As discussed in Section 4.1, the first sub-gene set represents job-position assignment (i.e., $g(n, j, 1) = p$) and the second one denotes job-configuration selection (i.e., $g(n, j, 2) = f$), where p is the job position and f is the configuration selected for job (c, j). Now if a data array is used to represent a parameter in the equations, then a parameter with one superscript and two subscripts can be denoted as a three dimensional array. For example, parameter UMI_{jf}^{c} can be expressed as $UMI(c, j, f)$. Replacing the configuration symbol f with its gene representation, $UMI(c, j, f)$ can be further incorporated in the program as $UMI(c, j, g(n, j, 2))$. Following this concept, the total machine idle cost in Equation (5) and the material handling cost in Equation (7) can be rewritten as (a) and (b) respectively in the following:

$$U_h * Q(c,j) * UMI(c,j,g(n,j,2)) * UPT(c,j,g(n,j,2)) \quad \text{(a)}$$

$$U_m * Q(c,j) * UMH(c,j,g(n,j,2)) \quad \text{(b)}$$

The first two summations of in the Equations (5) and (7) are simply done by looping over the indices of c, and j in the GA program. The third summation of has been embedded in the program. Variable Z_{jf}^{c} in the Equations (5) and (7) is the output.

System reconfiguration cost is dependent on the job sequences and the configurations selected for processing jobs. Hence, the implementation of system reconfiguration cost in the GA involves two steps: the determination of job sequence and the selection of configurations for two consecutive jobs in the sequence. Job sequence is represented by the first sub-gene set whose third array dimension is 1. For example, $g(n, i, 1)$ and $g(n, j, 1)$ denote the sequence positions of job (c, i) and (c, j) respectively. If $g(n, i, 1) = 2$ and $g(n, j, 1) = 3$, it means job (c, i) is in the second sequence position and job (c, j) immediately follows job (c, i) (i.e., sequence position 3). The configurations selected for these two jobs are represented by their corresponding sub-gene sets whose third array dimension is 2, i.e., $g(n, i, 2)$ and $g(n, j, 2)$ respectively. In this arrangement,

reconfiguration cost $CF^c_{f_j f_k}$ in Equation (3) can be expressed in the array format of $CF(c, f_1, f_2)$, where f_1 and f_2 are the configurations selected for processing two consecutive jobs in the sequence. The gene representations of f_1 and f_2 in the GA program are $g(n, i, 2)$ and $g(n, j, 2)$ respectively. The jobs using configurations f_1 and f_2 are their counterparts in the first sub-gene set which are $g(n, i, 1)$ and $g(n, j, 1)$ respectively. Hence $CF(c, f_1, f_2)$ can be rewritten as $CF(c, g(n,i,2), g(n, j,2))$. The summations in Equation (3) are accomplished by the looping operations on indices of c, p, j, and k in the GA program. Variables Y^c_{jpk}, $Z^c_{jf_j}$ and $Z^c_{kf_k}$ are the outputs of the program.

Incorporating the WIP holding cost in the objective function is described as follows.

(1) Substitute the job position p and configuration f with their gene representations of $g(n, j, 1)$ and $g(n, j, 2)$, Equation (8) is expressed as

$$ProcessTime(c, j, g(n, j, 1)) = Q(c, j) * UPT(c, j, g(n, j, 2)) \quad (c)$$

(2) Calculate the completion time of each job in a cell. If a job is assigned as the first job in the sequence in a cell, then the completion time of this job equals its processing time, i.e.,

$$CompletionTime(c, j) = ProcessTime(c, j, 1) \quad (d)$$

If a job is put in any other positions, then its completion time is the total processing time of this job and all other jobs that are arranged before it:

$$CompletionTime(c, j) = PrecedeJobs + ProcessTime(c, j, g(n, j, 1)) \quad (e)$$

(3) Recall that the job number j can be interpreted as the product variant number (refer to Table 3.3). The completion time of last job for product variant j is the maximum value among all $CompletionTime(c, j)$ where cell index c varies from 1 to the total cell number. Record the completion time of last job in variable $BuffTime(1, j)$.

(4) Calculate the completion time difference among the last job and all other jobs for product variant j. Store them in data array $DeltTime(c, j)$, $\forall c$.

(5) The WIP holding cost for product variant j is the product of unit holding cost and $DeltTime(c, j)$, $\forall c$.

(6) Repeat the same steps from (2) to (5) for other product variants. The total WIP is then obtained.

All cost components are incorporated in the objective function, which is also used for evaluating the fitness value. Since the goal is to minimize the overall production cost, chromosome with smaller fitness value is considered better one.

4.3.6 Decide the Stopping Criteria

Three criteria are used to decide when to halt the GA search process: maximum allowable number of generations in modifying the population, maximum allowable computing time and maximum allowable consecutive iterations (set to 100 in this study) that are unsuccessful in improving the performance (fitness function). Users can choose either of the first or second criterion. The third criterion has been built in the program. The search is stopped if the chosen criteria (the first or second one) is met or the maximum allowable consecutive iterations (the third criterion) is reached. The computing time criterion is mainly used for extremely large sized problems. In general, the choice of stopping rule is a tradeoff between convergence to optimality and execution time.

4.4 Flow Chart and Overall Procedure

Figure 4.6 shows the flow chart of the GA program. The temporary population contains all child chromosomes generated by crossover operation in each iteration. Its size is the same as the original population size. The fitness values of chromosomes in temporary population are then compared with those of the previous ones. Chromosomes with smaller fitness values have more chance to survive.

The GA algorithm incorporating the above coding scheme is summarized as follows:

1. Generate initial population and evaluate fitness value of each chromosome. The chromosome with the smallest fitness value is the current best candidate (i.e., the best chromosome obtained so far).
2. Choose two chromosomes from the population pool randomly or using roulette wheel selection.
3. Perform crossover operation on the selected chromosomes to generate a pair of offspring. Allocate them into a temporary population pool. If the predefined population size is not reached, go back to step 2.

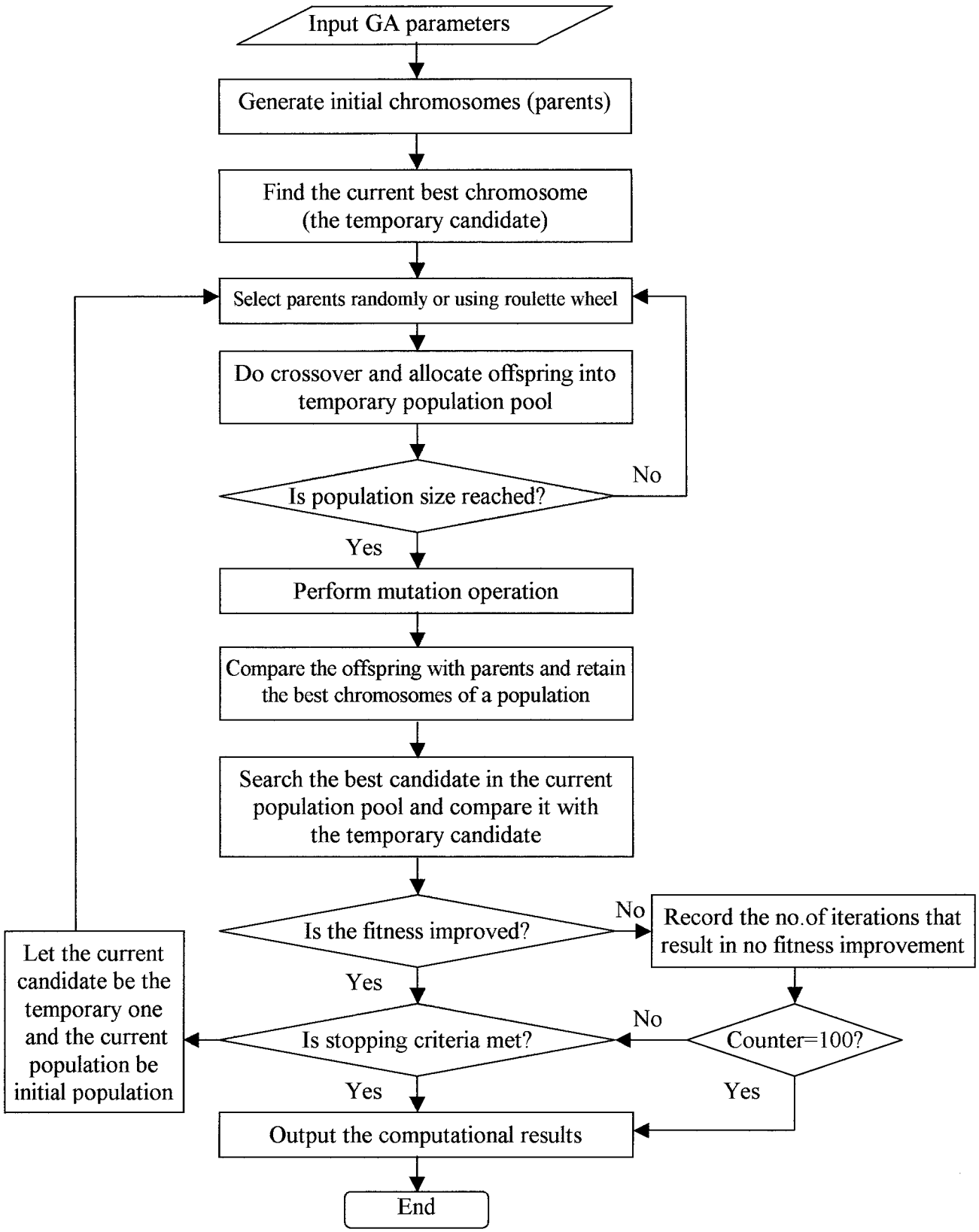


Figure 4.6 The flow chart of the GA program

4. Mutate the randomly selected chromosomes from the previous population until the specified number of chromosomes to mutate i.e., MutNum (refer to Section 4.3.4) is reached. Offspring with smaller fitness values will replace their corresponding parents.
5. Compare each offspring in the temporary population with chromosomes in the previous population. If improved, the offspring will survive. This operation generates a new population.
6. Search the best candidate in the new population and compare it with the temporary candidate. The current candidate is the one with smaller fitness value. Terminate the search process and adopt the current candidate as a final solution if any of the three stopping conditions is met. Otherwise, let the current candidate be the temporary candidate and proceed to the next iteration and return to step 2.

4.5 The Interfaces of the GA program

The GA program is coded using Visual Basic 6.0. There are three forms and one main module in this program and the detailed codes are listed in Appendix B. Running the GA program involves three basic steps: Input data, Select GA parameters, and Interpret the computational results.

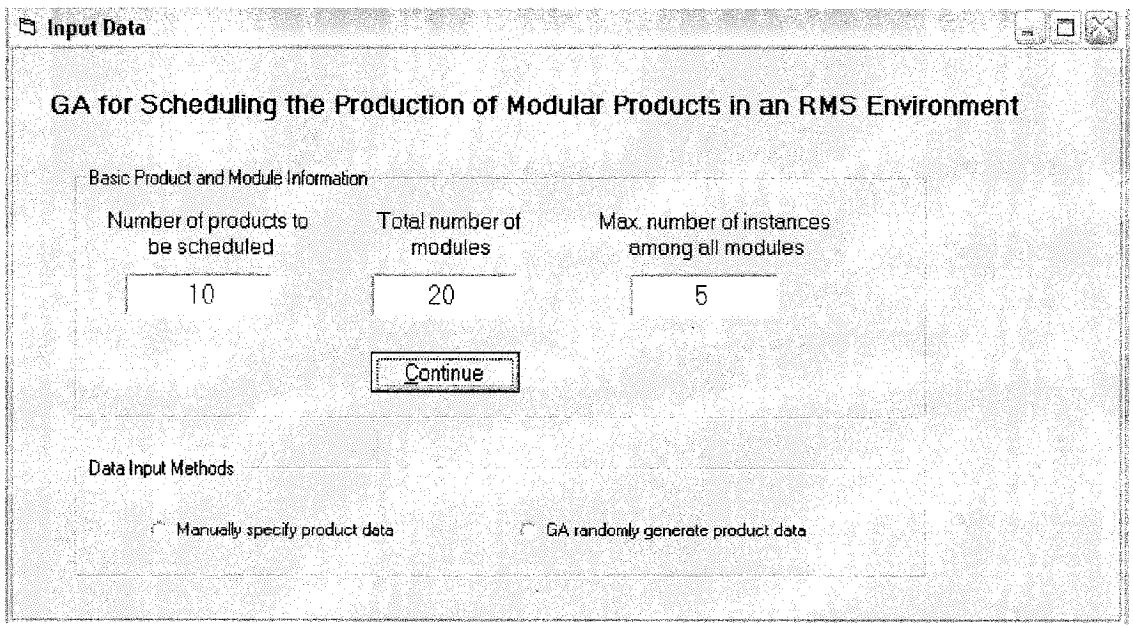


Figure 4.7 The input form of the GA program

(1) The first step is to input the number of product variants to be scheduled, the total number of modules and the maximum number of instances among all modules. Figure 4.7 is the input form that shows a problem with 10 product variants, each of which has 20 modules and the configuration number in each cell is 5. There are two options for loading the product and system information: load the data from a prepared file or let the GA program automatically generate the necessary data.

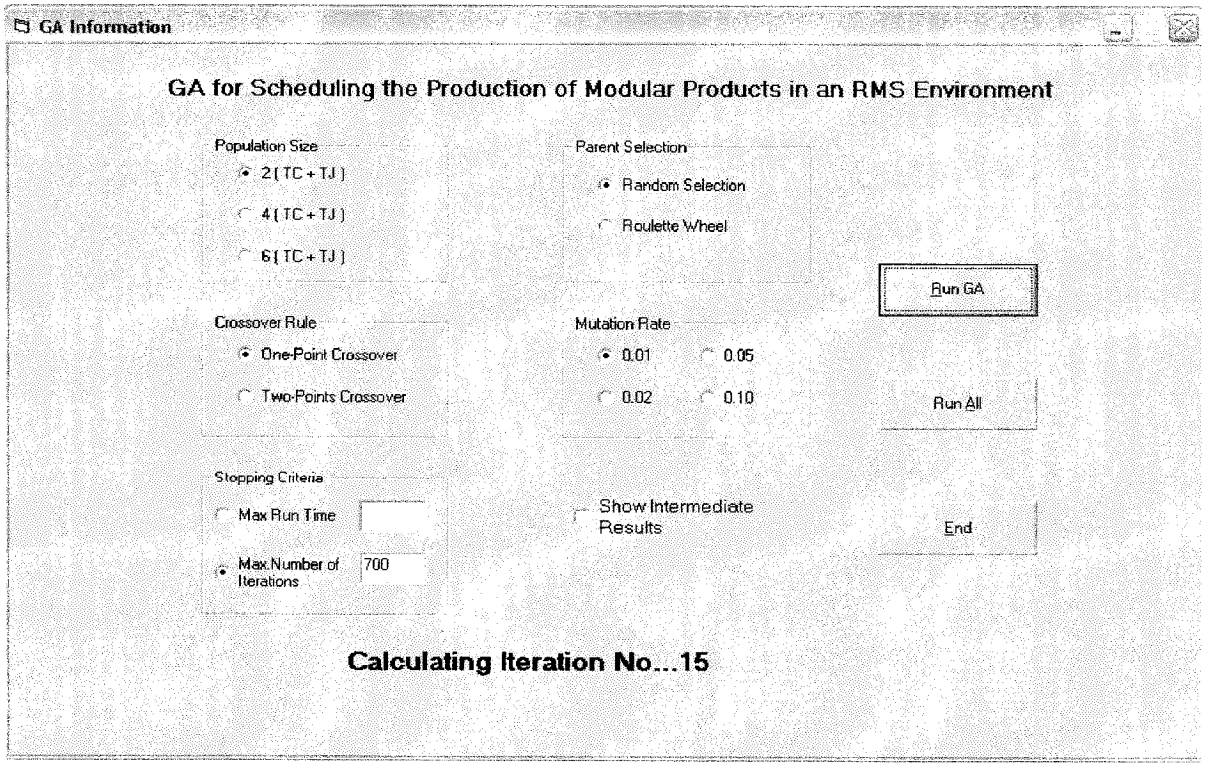


Figure 4.8 The GA information form

(2) The second step is to select a set of GA parameters to run the program. These parameters include the population size, parent selection, crossover rule, mutation rate. There are different combinations of these parameters, but the program ensures that only one set of parameters is effective at a time unless the user presses the "Run All" option which allows the program to run all the combinations one by one and output the best objective value. Note that only two stopping rules, i.e., maximum allowable number of generations, and maximum allowable computing time are shown in Figure 4.8. The third one, i.e., maximum allowable consecutive iterations has been set to 100 in the GA

program and is not shown in Figure 4.8. Users can choose either of the first two stopping rules, but the program will be ended in each case if there is no improvement on the objective value after 100 consecutive runs. For example, the maximum number of iterations (700) is selected as the stopping criterion in Figure 4.8, the program will iterate to this number if the objective value improves continuously, or stop earlier if the objective value remains the same after 100 consecutive iterations.

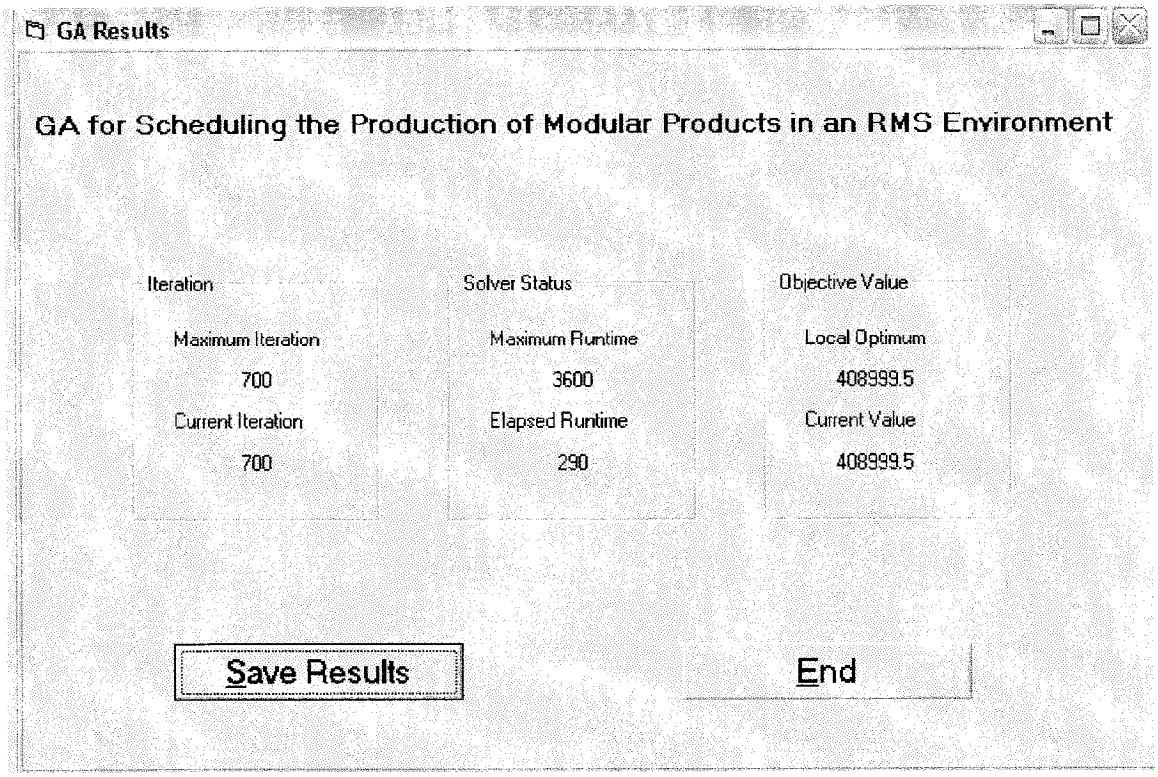


Figure 4.9 The result form of the GA program

(3) The output form, as shown in Figure 4.9, gives the iteration number, the elapsed run time and the objective value. Once the <Save Results> is pressed, the program will write the results to a text file.

4.6 Implementation of LINGO Program

LINGO is an optimization tool which can be utilized to solve both linear and nonlinear problems. Since it introduces the concept of "sets", LINGO formulates large problem concisely.

Sets are simply groups of related objects. Each member in the set may have one or more characteristics associated with it. These characteristics are called attributes. Attribute values can be known in advance or unknowns that LINGO solves. For example, each product might have a volume attribute and each machine might have a manufacturing capacity attribute, etc.

LINGO recognizes two kinds of sets: primitive and derived.

A primitive set is a set composed only of objects that cannot be further reduced. Four primitive sets are defined in this study: Product Variant (*VAR*), Job Position (*POS*), Configuration (*CON*), and Manufacturing Cell (*CEL*). Using the example in Table 3.2, the set of *VAR* is composed of 3 product variants and the set of *CEL* is formed by 3 manufacturing cells.

A derived set is defined using one or more other sets. In other words, a derived set derives its members from other existing sets. There are six derived sets in this LINGO model:

(a) *CEL_VAR* (*CEL, VAR*): *Q, CT, DELT*

This derived set has three attributes:

(1) $Q(c, j)$ is the job volume in each cell.

(2) $CT(c, j)$ is the job completion time in each cell.

(3) $DELT(c, j)$ is the completion time difference of product variant j in each cell.

(b) *VAR_POS* (*CEL, VAR, POS*): *X, CTP*

This derived set has two attributes:

(1) $X(c, j, p)$ is the job-position assignment variable, $X(c, j, p) = 1$ if job (c, j) is assigned to sequence position p in cell c; otherwise, 0.

(2) $CTP(c, j, p)$ is the completion time of job (c, j) if it is in position p in cell c.

(c) *VAR_POS_VAR* (*VAR_POS, VAR*)|*&4 #NE# &2 #AND# &3 #LT# @SIZE(POS)*: *Y*

This derived set has only one attribute: $Y(c, j, p, k) = 1$, if job (c, j) is assigned to position p and immediately followed by job (c, k) (k is not equal to j) in cell c; otherwise, 0

This derived set comes with membership filter to eliminate those members that are not needed in the set. The start of the membership filter is denoted with the vertical

bar character (|). The "&1" and "&2" symbols in the filter are known as set index placeholders. The logic comparators are included in the two #'s sign.

(d) *VAR_CON (CEL, VAR, CON): Z, FEA, UMH, UMI, UPT, CTF*

This derived set has six attributes:

(1) $Z(c, j, f)$= 1 if job (c, j) uses configuration f in cell c; otherwise,0.

(2) $FEA(c, j, f)$=1 if configuration f is feasible for job (c, j) in cell c; otherwise,0.

(3) $UMH(c, j, f)$ is the distance that a component has to travel, if configuration f is used to process job (c, j) in cell c.

(4) $UMI(c, j, f)$ is the number of idle machines during processing one unit of job (c, j) if configuration f is used in cell c.

(5) $UPT(c, j, f)$ is the number of time units required to process one unit of job (c, j) if configuration f is used in cell c.

(6) $CTF(c, j, f)$ is the completion time of job (c, j) if configuration f is used to process job (c, j) in cell c.

(e) *VAR_VAR_COST (CEL, VAR, CON, VAR, CON)|&4 #NE# &2: JFKF*

This derived set has only one attribute: $JFKF(c, j, f_j, k, f_k)$ which is the reconfiguration cost of changing system from job (c, j) to (c, k) (k not equal to j), with job (c, j) using configuarion f_j and job (c,k) using configuration f_k.

(f) *CON_CON (CEL, CON, CON): FFC*

This derived set has only one attribute: $FFC(c, f_1,f_2)$ which is the reconfiguration cost from cell configuration f_1 to f_2 in cell c.

Once the necessary sets are defined, the next step is to apply the set looping functions to all members of a set using a single statement. Set looping functions allow iterating through all the members of a set to perform some operation. The detailed illustrations of the use of set looping functions are given in the LINGO program (see appendix A).

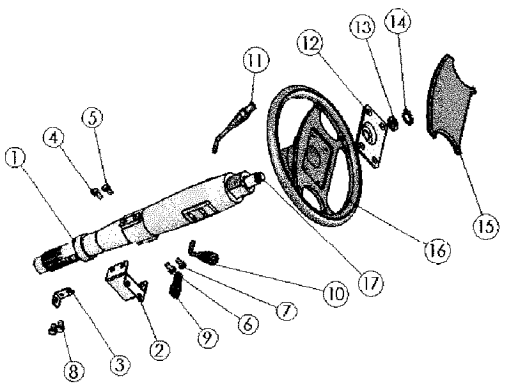
Chapter 5

Case Study and Computational Results

In this chapter, an industrial case is used to illustrate the implementation of the proposed model and its GA solution procedure. The computational results are analyzed using ANOVA(analysis of variance) method and the effects of GA parameters are discussed.

5.1 Case Description

To apply the proposed GA method, a manufacturing problem of a steering column from the automotive industry is selected. The steering column is responsible for transmitting the control of a moving vehicle from the steering wheel to the steering gear box. This case is retrieved from the example provided by Rekiek *et al* (1997).



1	Steering Column	10	Turn Lever
2	Mount Bracket	11	Cruise Lever
3	Sensor Bracket	12	Damper
4	Bolt #1	13	Steering Wheel Nut
5	Bolt #2	14	Retainer
6	Bolt #3	15	Horn Pad
7	Bolt #4	16	Steering Wheel
8	Bolt #5	17	Connecting Pin
9	Tilt Handle		

Figure 5.1 An exploded view of steering column and its bill of material

A manufacturer is to produce three different models of a steering column in a reconfigurable manufacturing system. These three models have most of their parts in common, but options include a turn/cruise lever rather than the basic turn lever, a hazard switch, damper and a tilt option on the steering wheel. The production ratio for each variant is respectively 63%, 29% and 8%. The total number of products to schedule is 250. An exploded view of the steering column and its bill of material are illustrated in Figure 5.1. There are a total of 17 parts in this steering column assembly. Manufacturing

Table 5.1 Manufacturing task distributions

Cell	Task No.	Task Description	Operation Time(sec)	Machine Relocation Cost ($)
1	1	Surface turn upper end and thread	68	400 (80,320)
	2	Center drill upper end for connecting pin	22	280 (56,224)
	3	Surface turn lower end and cut pinion	185	880 (176,704)
	4	Mill surface and tap holes for mount bracket	25	150 (30,120)
	5	Mill surface and tap holes for sensor bracket	25	150 (30,120)
	6	Slot hole for turn lever	35	180 (36,144)
	7	Slot hole for cruise lever	35	180 (36,144)
	8	Slot hole for tilt lever	30	180 (36,144)
	9	Powder coating	45	1500(300,1200)
2	1	Laser-cut sheet metal for mount bracket	10	350 (70,280)
	2	Bend flanges of mount bracket	58	450 (90,360)
	3	Saw-cut angle bar for sensor bracket	15	240 (48,192)
	4	Slot holes on sensor bracket	10	140 (28,112)
	5	Saw-cut turn lever rod to length	15	240(48,192)
	6	Bend turn lever rod to angle	28	450 (90,360)
	7	Press fit plastic handle on turn lever rod	8	100 (24,96)
	8	Saw-cut cruise lever rod to length	15	240(48,192)
	9	Bend cruise lever rod to angle	28	450 (90,360)
	10	Press fit plastic handle on cruise lever rod	8	100 (24,96)
	11	Saw-cut tilt handle bar to length	15	240(48,192)
	12	End pocket tilt handle bar for tilt button	30	240(48,192)
	13	Sand blasted tilt handle	20	600(120,480)
3	1	End surfacing on hub of wheel	20	280(56,224)
	2	Bore center hole	47	250 (50,200)
	3	Pocket to accept horn button	85	120 (24,96)
	4	Taper button hole in the hub of wheel	60	190 (38,152)
	5	Small slot in pocket for horn wire	12	140 (28,112)
	6	Drill and tap holes for horn pad	22	200(40,160)
	7	Powder coating black color	64	1500 (300,1200)
	8	Powder coating brown color	64	1500 (300,1200)
	9	Powder coating beige color	64	1500 (300,1200)
4	1	Laser cut the profile of horn pad	26	350 (70,280)
	2	Stamping horn pad to accept air bag	22	680 (128,552)
	3	Stud weld for mounting horn pad	32	250 (50,200)
	4	Hand polish	49	100(20,80)
	5	Leathering horn pad	79	280(56,224)
	6	Bore center hole on damper	21	280(56,224)
	7	Drill mounting holes on damper	27	200(40,160)
	8	Sand blasted damper	42	800(160,640)

Note: The numbers inside parentheses in column 5 are machine removal costs and arrangement costs respectively.

parts include the steering column, mount bracket and sensor bracket, tilt handle, turn/cruise lever, steering wheel, damper and horn pad. Table 5.1 summarizes the basic information of the manufacturing processes of these components. It describes the tasks distributed among four manufacturing cells, and gives the processing time for each manufacturing operation in a cell. Cell 1 is mainly used to produce the steering column. The mounting bracket, sensor bracket, turn lever, cruise lever and tilt handle are completed in cell 2. Cell 3 is responsible for processing the steering wheel. Horn pad and damper are finished in cell 4. The estimated processing time of each machine and it relocation cost are also given in Table 5.1

Table 5.2 shows the model (product variant) differences by giving the specific task requirements of each model. As shown, Model 1 has the options including a damper, a turn/cruise lever, a hazard switch, and a tilt level, and accounts for 63% of the annual demand. Model 2 is the basic model with no options, and accounts for 29% of demand. Model 3 has a damper and a turn/cruise lever, but no tilt lever and hazard switch, and accounts for 8% of the total demand.

Table 5.2 The task differences among three product variants

Product Variant	Operation tasks distributed among manufacturing cells			
	Cell 1	Cell 2	Cell 3	Cell 4
Model 1	1,2,3,4,5,6,7,8,9	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7	1,2,3,4,5,6,7,8
Model 2	1,2,3,4,6,9	1,2,5,6,7	1,2,3,4,5,6,8	1,2,3,4,5
Model 3	1,2,3,4,6,7,9	1,2,5,6,7,8,9,10	1,2,3,4,5,6,9	1,2,3,4,5,6,7,8

To facilitate the discussion in the following sections, the module instances among the four manufacturing cells are defined as follows:

Product variant 1 needs 9 operations to complete in cell 1 (refer to Table 5.2), hence the first instance of module 1 (M_{11}) contains 9 operations. Product variant 3 has 7 processing operations in cell 1, and they are defined as the second instance of module 1 (M_{12}). The third instance of module 1 (M_{13}) consists of 6 operations needed for processing product variant 2 in cell 1.

Table 5.3 Module components of the three steering columns

Product Variant	M_1			M_2			M_3			M_4	
	M_{11}	M_{12}	M_{13}	M_{21}	M_{22}	M_{23}	M_{31}	M_{32}	M_{33}	M_{41}	M_{42}
1	Δ			Δ			Δ			Δ	
2			Δ			Δ		Δ			Δ
3		Δ			Δ				Δ	Δ	

In cell 2, product variant 1 requires all operations (M_{21}) and product variant 3 has 8 operations (M_{22}). Only 5 operations are needed for producing product variant 2 in cell 2 (M_{23}).

For cell 3, all 3 product variants shares most of manufacturing resources, i.e., machines 1 to 6. In the last operation, product variants 1,2 and 3 are painted in brown color (M_{31}), black color (M_{32}) and beige color (M_{33}) respectively.

In cell 4, both of product variants 1 and 3 have the options of a damper and horn pad (M_{41}), product variant 2 is the basic model (M_{42}).

The module instances contained in these three product variants are summarized in Table 5.3. There are a total of 4 modules. Each of modules 1 to 3 has 3 instances and module 4 contains only 2 instances.

Table 5.4 Jobs distributed in four cells

Product variant	Components of Product Variant	Cell 1		Cell 2		Cell 3		Cell 4	
		Job	Inst	Job	Inst	Job	Inst	Job	Inst
1	$M_{11}+M_{21}+M_{31}+M_{41}$	J_{11}	M_{11}	J_{21}	M_{21}	J_{31}	M_{31}	J_{41}	M_{41}
2	$M_{13}+M_{23}+M_{32}+M_{42}$	J_{12}	M_{13}	J_{22}	M_{23}	J_{32}	M_{32}	J_{42}	M_{42}
3	$M_{12}+M_{22}+M_{33}+M_{41}$	J_{13}	M_{12}	J_{23}	M_{22}	J_{33}	M_{33}	J_{43}	M_{41}

Following the procedures discussed in Section 3.1, the next step is to distribute the production of these three product variants among the four manufacturing cells. Consider the production of different module instances in cell 1. There are three batches of jobs: job 1 in cell 1 (J_{11}) contains D_1 pieces of M_{11}, job 2 in cell 1 (J_{12}) has D_2 pieces of M_{13}, and job 3 in cell 1 (J_{13}) contains D_3 pieces of M_{12}, where D_1, D_2 and D_3 are the demands of

product variants 1, 2 and 3 respectively. The job details in the four cells are shown in Table 5.4.

5.2 Cell Configurations

The module instances included in each product variant have been defined in Section 5.1. Correspondingly, the cell configuration used to produce each module instance is explained in this section.

Each of cells 1, 2 and 3 has three configurations, and each configuration is responsible for processing an instance of its corresponding module. Since module 3 has only two instances, there are two configurations in cell 4. The operation sequence in a configuration, and unit processing time for each configuration in a cell are listed in Table 5.5. The unit processing time of each configuration is obtained by summing up the operation times of all machines involved in a configuration. For example, the operation times of machines 1, 2, 3, 4, 5, 6, 7, 8 and 9 in configuration f_1 of cell 1 are 26,22,32,49 and 79 seconds respectively (Table 5.1). The unit processing time of configuration f_1 of cell 1 is the sum of the operation times of these 9 machines, i.e., 465 seconds. The unit processing times for other configurations can be calculated similarly.

Table 5.5 Cell configurations and unit processing time

Cell	Configuration	Operation sequence	Unit processing time
1	f_1	1,2,3,4,5,6,7,8,9	465
	f_2	1,2,3,4,6,7,9	415
	f_3	1,2,3,4,6,9	380
2	f_1	1,2,3,4,5,6,7,8,9,10,11,12,13	250
	f_2	1,2, 5,6,7,8,9,10	170
	f_3	1,2,5,6,7	119
3	f_1	1,2,3,4,5,6,7	308
	f_2	1,2,3,4,5,6,8	312
	f_3	1,2,3,4,5,6,9	310
4	f_1	1,2,3,4,5,6,7,8	298
	f_2	1,2,3,4,5	208
	f_3	Dummy	10000

A dummy configuration f_3 is added in cell 4 to make the number of configurations in all cells be the same, in this case, three. The dummy configuration is used to facilitate the looping operation in the proposed GA program so that the same program module can be

used for each cell. To prevent the dummy configuration from being selected for processing a job, its unit processing time is given a high value of 10000.

5.3 Job-Configuration Feasibility Matrix

Once the descriptions of the jobs and the definitions of cell configurations are given, the next step is to find out the job-configuration feasibility matrix.

As mentioned in Section 5.2, each of cells 1, 2, and 3 has three configurations Obviously, configuration f_1 in cell 1 is capable of processing any instances of module 1 and hence it is feasible for all jobs in cell 1. Configuration f_2 can be used for processing J_{12} and J_{13}. Configuration f_3 can only be used to produce J_{12}. The job-configuration feasibility matrix in cell 1 is shown under the heading of FEA in Table 5.6.

Table 5.6 Input data for cell 1

Job	FEA			UMH			UMI			UPT		
	f_1	f_2	f_3	f_1	f_2	f_3	f_1	f_2	f_3	f_1	f_2	f_3
J_{11}	1	0	0	8	BM	BM	0	BM	BM	465	BM	BM
J_{12}	1	1	1	8	6	5	3	2	0	380	380	380
J_{13}	1	1	0	8	6	BM	2	0	BM	415	415	BM

Note: 1. FEA is the configuration-job feasibility matrix, f_j=1(j=1,2 and 3) means the configuration is feasible for its corresponding job, otherwise, 0.
2. UMH is the total number of idle machines if configuration f_j (j=1,2 and 3) is used to process one unit of J_{cj} (c=1, j=1,2 and 3).
3. UMI is the total units of machine distance that a component has to travel if configuration f_j (j=1,2 and 3) is used to process one unit of J_{cj} (c=1, j=1,2 and 3).
4. UPT is the unit job processing time if configuration f_j (j=1,2 and 3) is used.
5. BM is a large positive number(10000 units in this case) imposed on UMH, UMI or UPT if configuration f_j (j=1,2 and 3) is infeasible for J_{cj} (c=1, j=1,2 and 3), or if dummy configuration f_3 in cell 4 is selected.

Table 5.7 Input data for cell 2

Job	FEA			UMH			UMI			UPT		
	f_1	f_2	f_3	f_1	f_2	f_3	f_1	f_2	f_3	f_1	f_2	f_3
J_{21}	1	0	0	12	BM	BM	0	BM	BM	250	BM	BM
J_{22}	1	1	1	12	7	4	8	3	0	119	119	119
J_{23}	1	1	0	12	7	BM	5	0	BM	170	170	BM

Similarly, the job-configuration feasibility matrices in cell 2 can be determined. Configuration f_1 is capable of processing all module instances in cell 2. Configuration f_2 can be used for processing either J_{22} or J_{23}. Configuration f_3 is only feasible for J_{22}. The feasibility matrices in cell 2 is shown under the headings of FEA in Table 5.7.

Table 5.8 Input data for cell 3

Job	FEA			UMH			UMI			UPT		
	f_1	f_2	f_3	f_1	f_2	f_3	f_1	f_2	f_3	f_1	f_2	f_3
J_{31}	0	0	0	6	BM	BM	0	BM	BM	308	BM	BM
J_{32}	1	1	0	BM	6	BM	BM	0	BM	BM	312	BM
J_{33}	0	0	1	BM	BM	6	BM	BM	0	BM	BM	310

In cell 3, the only difference among three configurations is that the steering wheel is painted in different colors in the last operation. Hence each configuration is only feasible for its corresponding job, i.e., f_1 for black, f_2 for beige, and f_3 for brown color. The job-configuration feasibility matrix in cell 3 is shown under the heading of FEA in Table 5.8.

Table 5.9 Input data for cell 4

Job	FEA			UMH			UMI			UPT		
	f_1	f_2	f_3	f_1	f_2	f_3	f_1	f_2	f_3	f_1	f_2	f_3
J_{41}	1	0	0	7	BM	BM	0	BM	BM	298	BM	BM
J_{42}	1	1	0	7	4	BM	3	0	BM	208	208	BM
J_{43}	1	0	0	7	BM	BM	0	BM	BM	298	BM	BM

In cell 4, there are only two configurations in cell 4. Configuration f_1 is feasible for all jobs and configuration f_2 is solely used for J_{42}. The dummy configuration f_3 is infeasible to any instances in cell 4. The job-configuration feasibility matrix in cell 4 is shown under the heading of FEA in Table 5.9.

5.4 Formatting the Input Data

To implement the proposed GA method, different cost elements, i.e., the number of idle machines, number of component traveling distances, unit processing times, and WIP holding cost, should be defined. The input data are formatted in the following sections.

5.4.1 Component Travel Distance

Based on the information provided in Table 5.2 and Table 5.5, the number of units of component traveling distance in a cell configuration selected for a job can be determined. For example, if configuration number 2 is selected to process job 1 in cell 1, each component will travel 8 units of machine distance. The following should be considered in determining the component travel distance:

(1) To prevent the dummy configuration (f_3) in cell 4 from being selected to process a job, a penalty cost of BM=10000 units of component travel distance is imposed on it.

(2) If a configuration is not feasible for a job, BM=1000 units of component travel distance is introduced.

The component travel distances in a configuration selected for processing a job in cells 1 to 4 are listed under the heading of UMH from Table 5.6 to Table 5.9 respectively.

5.4.2 Number of Idle Machines

The number of idle machines in a cell configuration selected for a job can also be determined from Table 5.2 and Table 5.5. For example, if configuration f_1 is selected to process job 2 in cell 1 (J_{12}), machines 5, 7 and 8 will be idle-running. It is important to point out the following:

(1) To prevent the dummy configuration (f_3) in cell 4 from being selected to process a job, a penalty cost of BM=10000 units of idle machines is imposed on it.

(2) If a configuration is not feasible for a job, BM=10000 units of idle machines are introduced.

The numbers of idle machines in a configuration selected for a job in cells 1 to 4 are given under the heading of UMI from Table 5.6 to Table 5.9 respectively.

5.4.3 Unit Processing Time

As discussed in Section 3.3.1, there is a most appropriate configuration for each instance. The unit processing time of a job is the sum of operation times of all machines in the configuration selected for that job. For example, if configuration f_2 is employed for processing job 1 in cell 1 (J_{12}), the unit processing time will be 68+22+185+25+35+45=380 seconds, where 68, 22, 185, 25, 35 and 45 are the operation

times of machines 1, 2, 3, 4, 6, and 9 respectively (Table 5.1). If a configuration is infeasible for a job, the unit processing time of this job using this configuration will be replaced by BM (big positive number). The criterion to decide the value of a BM is to prevent this configuration from being selected for the job. For example, f_2 in cell 1 is not feasible for processing J_{13}, so the unit processing time of J_{13} using f_1 is BM. The value of BM is set at 10000 in this case. The unit processing time of each instance using different configurations in cells 1 to 4 is shown under the heading of UPT in Table 5.6 to Table 5.9 respectively.

5.4.4 Cell Reconfiguration Costs

The machine relocation costs need to be determined to facilitate the calculation of cell reconfiguration costs. As discussed in Section 3.3.1, the relocation cost of a machine consists of removal and arrangement costs. The removal and arrangement costs of all machines are shown in the last column in Table 5.1.

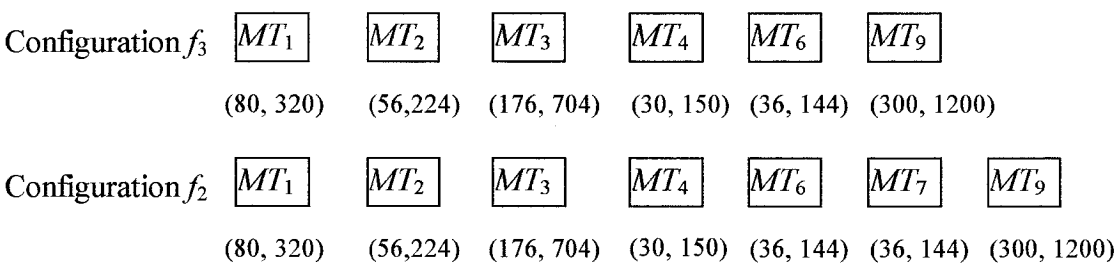


Figure 5.2 Machine removal and arrangement costs

The cell reconfiguration cost between any two configurations can be calculated based on the removal and arrangement costs of machines. Consider configurations f_2 and f_3 in cell 1 as shown in Figure 5.2 (the numbers below each machine block in the figure are respectively removal cost and arrangement cost). Configuration f_3 will be modified to f_2 by relocating machine 9 (costs 300 for removal and 1200 for arrangement) and adding machine 7 (costs 144 for arrangement). The cost of reconfiguring systems from f_3 to f_2 is therefore 300+1200+144=1644. Similarly, removing machines 7 (cost 36) and relocating machine 9 to the last position (involving both removal and arrangement at an expense of 300+1200=1500) in the machine sequence will change configuration f_2 to f_3. However, the reconfiguration cost is 36+300+1200=1536. Note that there is a dummy configuration

Table 5.10a Calculating the reconfiguration costs

Cell	Reconfiguration costs
1	$CF^1_{f_1 f_2} = RM_5 + RL_6 + RL_7 + RM_8 + RL_9 = 30+180+180+36+1500 = 1926$
	$CF^1_{f_2 f_1} = RA_5 + RL_6 + RL_7 + RA_8 + RL_9 = 150+180+180+144+1500 = 2154$
	$CF^1_{f_1 f_3} = RM_5 + RL_6 + RM_7 + RM_8 + RL_9 = 30+180+36+36+1500 = 1782$
	$CF^1_{f_3 f_1} = RA_5 + RL_6 + RA_7 + RA_8 + RL_9 = 150+180+144+144+1500 = 2118$
	$CF^1_{f_2 f_3} = RM_7 + RL_9 = 36+1500 = 1536$
	$CF^1_{f_3 f_2} = RA_7 + RL_9 = 144+1500 = 1644$
2	$CF^2_{f_1 f_2} = RM_3 + RM_4 + (RL_5 \rightarrow RL_{10}) + RM_{11} + RM_{12} + RM_{13}$ $= 48+28+240+450+100+240+450+100+48+40 = 1744$
	$CF^2_{f_2 f_1} = RA_3 + RA_4 + (RL_5 \rightarrow RL_{10}) + RA_{11} + RA_{12} + RA_{13}$ $= 192+112+240+450+100+240+450+100+192+160+480 = 2716$
	$CF^2_{f_1 f_3} = RM_3 + RM_4 + (RL_5 \rightarrow RL_7) + RM_8 + RM_{13}$ $= 48+28+240+450+100+48+90+20+48+40+120 = 1159$
	$CF^2_{f_3 f_1} = RA_3 + RA_4 + (RL_5 \rightarrow RL_7) + RA_8 + RA_{13}$ $= 192+112+240+450+100+192+360+80+192+160+480 = 2558$
	$CF^2_{f_2 f_3} = RM_8 + RM_9 + RM_{10} = 48+90+20 = 158$
	$CF^2_{f_3 f_2} = RA_8 + RA_9 + RA_{10} = 192+360+80 = 632$
3	$CF^3_{f_1 f_2} = RM_7 + RA_8 = 300+1200 = 1500$
	$CF^3_{f_2 f_1} = RA_7 + RM_8 = 300+1200 = 1500$
	$CF^3_{f_1 f_3} = RM_7 + RA_9 = 300+1200 = 1500$
	$CF^3_{f_3 f_1} = RA_7 + RM_9 = 300+1200 = 1500$
	$CF^3_{f_2 f_3} = RM_8 + RA_9 = 300+1200 = 1500$
	$CF^3_{f_3 f_2} = RA_8 + RM_9 = 300+1200 = 1500$
4	$CF^4_{f_1 f_2} = RM_6 + RM_7 + RM_8 = 56+40+160 = 256$
	$CF^4_{f_2 f_1} = RA_6 + RA_7 + RA_8 = 224+160+640 = 1024$
	$CF^4_{f_1 f_3} = CF^4_{f_3 f_1} = CF^4_{f_2 f_3} = CF^4_{f_3 f_2} = 10000$

Note: $CF^c_{f_j f_k}$ is the reconfiguration cost from f_j to f_k (j,k=1,2,3)in cell c, RM_s and RA_s are the removal cost and arrangement cost of machine s. $RL_s = RM_s + RA_s$ (s is the machine index).

f_3 in cell 4. The reconfiguration cost between dummy and f_2 or f_1 is simply given a large value (10000, in this case) to prevent the dummy configuration is selected for processing a job. Table 5.11a summarizes how to calculate the reconfiguration costs in each manufacturing cell.

Table 5.10b Reconfiguration cost (in $) matrix

Job	Cell 1			Cell 2			Cell 3			Cell 4		
	f_1	f_2	f_3	f_1	f_2	f_3	f_1	f_2	f_3	f_1	f_2	f_3
f_1	0	1926	1782	0	1744	1159	0	1500	1500	0	256	10000
f_2	2154	0	1536	2716	0	158	1500	0	1500	1024	0	10000
f_3	2118	1644	0	2558	632	0	1500	1500	0	10000	10000	0

Reconfiguration costs in other cells are calculated similarly. The reconfiguration costs in the five manufacturing cells are shown in Table 5.11b.

5.5 Computational Results

For comparison purpose, consider two batches of order. The first order is 250 units of product variants and the production ratio for each variant is respectively 63%, 29% and 8%. Another order is 5500 units of product variants and the percentage of each variant is randomly generated as 23%, 57% and 20% respectively.

Table 5.11 Results of job sequences and configurations selected for jobs

Case 1—Production of 250 units of 3 product variants (Production Ratio: 63%, 29% and 8%)				Case 2—Production of 5500 units of 3 product variants (Production Ratio: 23%, 57% and 20%)			
Cell 1	2[a]	3	1	Cell 1	2	3	1
	f_1[b]	f_1	f_1		f_1	f_3	f_2
Cell 2	2	3	1	Cell 2	3	2	1
	f_1	f_1	f_1		f_1	f_2	f_2
Cell 3	2	3	1	Cell 3	2	3	1
	f_1	f_2	f_3		f_1	f_2	f_3
Cell 4	2	3	1	Cell 4	2	3	1
	f_1	f_1	f_1		f_1	f_2	f_1

Note: [a] Product variant number; [b] cell configuration used for the associated product variant.

The proposed genetic algorithm is coded using Visual Basic 6.0 and is applied to the two cases of production scheduling problem. The outputs are summarized in Table 5.12.

The values in each cell represent the job sequence positions and the letter f with subscript denotes the configuration selected for its corresponding job. For instance, the job sequence in cell 1 of case 1 is $J_{12} \rightarrow J_{13} \rightarrow J_{11}$ and all these three jobs use the same configuration f_1.

For case 1, the job sequences in all cells are the same (i.e., job 2→job 3→job 1). Configuration f_1 is selected for all jobs throughout the planning horizon in cells 1, 2 and 4. Cell configuration needs to be reconfigured twice in cell 3(i.e., $f_1 \rightarrow f_2$ and $f_2 \rightarrow f_3$). For case 2, jobs in cells 1, 3 and 4 follow the same sequence (i.e., job 2→job 3→job 1) and each of these three cell is reconfigured twice. The job sequence in cell 2 is job 3→job 2→job 1 and only one cell reconfiguration is required.

5.6 Comparison of the Results of GA and LINGO

To evaluate the effectiveness of GA and the solution quality, the two cases provided in Section 5.5 are first solved using commercial software package LINGO8.0. The GA program and LINGO are run on an Intel Pentium M personal computer with a 1.5GHz Centrino processor and 512MB of RAM. The proposed GA uses a parallel chromosome coding scheme and the results are compared in Table 5.13.

Table 5.12 Computational results of LINGO and GA

Problem	LINGO 8.0		Proposed GA		Improvement Rate on Cost (%)
	Cost ($)	Time (sec)	Cost ($)	Time (sec)	
Case 1	16427.3	199	15531.4	2	5.6 %
Case 2	231045	311	230630.1	2	2 %

For case 1, the improvement rate of objective value is about 5.6%. Both LINGO and GA obtain almost the same objective value for case 2. However, the proposed GA approach takes much shorter time, only 2 seconds in each case, to reach a near-optimal solution while the computing times of LINGO are 199 and 311 seconds for case 1 and case 2 respectively.

Table 5.13 Data ranges of randomly generated input data

Demand	FFC	UMH	UMI	UPT
200~1000	20~100	2~50	1~20	5~50

In order to further test the efficiency of the proposed GA method, four small and six large size problems are generated. The initial input data are randomly generated within predefined ranges as shown in the Table 5.14. For example, the product demand is randomly generated between 200 and 1000. The outputs are summarized in Table 5.15.

Table 5.14 Computational results of small size models

Model Size	LINGO 8.0		GA with serial chromosome coding		GA with parallel chromosome coding	
	Objective Value	Time (hr.min.sec)	Objective Value	Time (sec)	Objective Value	Time (sec)
V3C4L3*	4825	0.07.03	4641.9	<1	4719.6	<1
V4C3L4	7107.7	0.47.35	5573.8	<1	5546.2	<1
V4C4L4	9155.7	2.37.33	8349.9	1	8659.8	1
V4C5L4	16147.5	5.31.21	11606.7	2	11056.3	2

*Note: V3C4L3 represents a problem with 3 product variants processed in 4 cells and the number of configurations is 3

As shown in Table 5.15, with LINGO the computation time increases dramatically as the model size grows. The number of combinations of job sequences and cell configurations can be estimated as:

$$TN = (\prod_{c=1}^{C} \prod_{j=1}^{N} TF_j^c)(N!)$$

where TF_j^c is the number of configurations included in the feasible set of configurations for job (c,j) in cell c and N is the total number of product variants. For the first problem V3C4L3 in Table 5.15, the number of possible combinations, TN, is over 6.88×10^8. If the model size changes to V4C5L4, i.e., an increase of 1 in product variant, cell, and number of configurations, TN will reach roughly 8.55×10^{15}. Obviously, TN increases exponentially with the model size. Since all variables are integer and Equation (19) contains the product of variables $Z_{jf_j}^c$ and $Z_{kf_k}^c$, this model addresses a non-linear integer problem. LINGO is a generic software package based on the branch-and-bound method in solving integer models. The branch-and-bound method is enumerative in nature and cannot solve such a hard combinatorial problem in polynomial time. Our computational experience also shows that LINGO stops search after locating the first local optimum. As shown in Table 5.15, with LINGO the computation time (the time to reach the first local optimum) increases dramatically as the model size grows.

The proposed GA can achieve better results within much shorter time for each model size. For example, GA achieves objective value of 11056.3 for model size V4C5L4, which is over 31% improvement of the objective value obtained by LINGO. The computing time of GA is 2 seconds while LINGO runs more than five and half hours to reach a local optimum. Obviously, LINGO is not effective for solving large size problems in terms of both solution quality and computing time.

Table 5.15 Computational results of large size models

Model Size	GA with serial chromosome coding		GA with parallel chromosome coding	
	Time (seconds)	Objective Value	Time (seconds)	Objective Value
V10C20L5	482	534508	428	405285
V10C30L5	1448	974373	800	900261
V10C20L30	2297	532218	800	319121
V20C10L5	1550	989167	800	492704
V20C20L30	3058	2034618	801	1959636
V20C30L5	3124	4481952	805	4072495

The proposed GA is further tested using larger problems with up to 20 products, 30 cells and 30 configurations as shown in Table 5.16. Problems of this size range should cover a wide range of industrial applications. Relatively small population is used in the tests to facilitate the search process. The proposed approach can find near-optimal solutions within 800 seconds, which is quite reasonable considering the nature of the problem.

5.7 Evaluation of the GA performance

Though GA is effective in solving the combined modular product scheduling and cell configuration selection problem, the results are, to a certain extent, affected by GA parameters. Therefore it is necessary to investigate the effects of GA parameters on search performance in terms of convergence behavior and solution quality.

5.7.1 GA Convergence

The problem of size V10C20L5 is used to further examine the performance of the proposed GA approach. The required input data are randomly generated by the computer within predefined range.

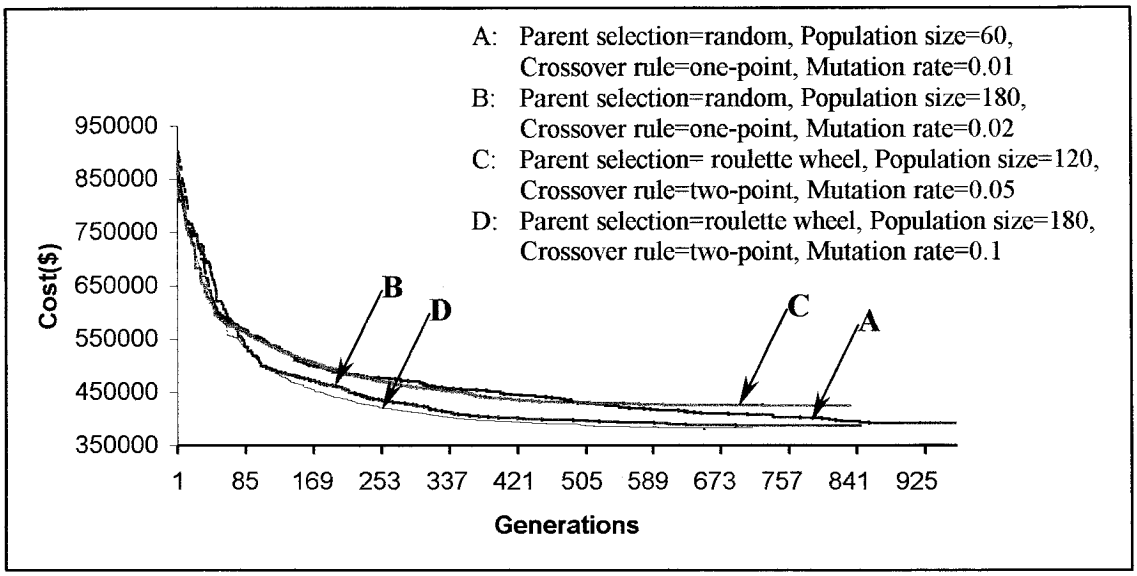


Figure 5.3 GA convergence

Figure 5.3 shows four plots of the cost objective values corresponding to four sets of GA parameters, namely parent selection method, population size, crossover rule and mutation rate. The termination condition is set to maximum runtime of one hour. There is no improvement of objective value for all cases after 1500 generations. It can be seen that GA reduces the objective value rapidly at the beginning of evolution. During the first 600 iterations, GA achieves about 92% of the cost reduction. Plot A is obtained by using random parent selection, population size of 60, one-point crossover and mutation rate of 0.01. It shows a relative slow convergence rate, but it achieves the best objective value of 379732.6. Plot B uses a population size of 180 and mutation rate of 0.02 while the other two parameters are the same as the ones used in plot A. It converges faster than plot A, but the objective value is 387439.2, which is nearly 2% higher than plot A. The minimal objective value (383744.4) of plot D is achieved by adopting roulette wheel selection, a population size of 180, two-point crossover and mutation rate of 0.1. It is almost the same as that of plot B. The convergence rates of plots D and B are very similar. Plot C employs a population size of 120 and mutation rate of 0.05 while other parameters remain the

same as those of plot D. It shows a similar convergence rate, but it reaches the objective value of 425108, which is about 10% higher than the best value achieved.

5.7.2 Effects of GA Parameters

Since the GA parameters are domain-specific, extensive preliminary experiments were carried out to determine ranges of appropriate parameter values. To further investigate how GA parameters affect the performance of the proposed approach, a multi-factor experiment is designed with the four GA parameters as factors as shown in the Table 5.17.

Table 5.16 GA parameters

Factors	Levels
(a) Parent selection	(1) Rule 1: Random selection (2) Rule 2: Roulette wheel selection
(b) Population size	(1) 2 (*TC*+*TJ*) (2) 4 (*TC*+*TJ*) (3) 6 (*TC*+*TJ*)
(c) Crossover rule	(1) Rule 1: One-point crossover (2) Rule 2: Two-point crossover
(d) Mutation rate	(1) 0.01 (2) 0.02 (3) 0.05 (4) 0.1

Note: *TC* - total number of manufacturing cell
TJ – total number of product variants to be scheduled

In this study, two parent selection methods, i.e., random and roulette wheel selections, are considered. For crossover rule, two policies, i.e., one and two cutting points, are introduced. The determinations of population size and mutation rate are based on computational experiments. Three levels of population sizes, 60,120 and 180, are selected and four levels of mutation rates, 0.01, 0.02, 0.05 and 0.1 are used. This requires multi-factorial ANOVA analysis to study the nature of any interaction that may be present between the four factors. ANOVA method is used to detect significant differences in each of the performance indices' mean values at different levels of the controlling factor. It is also used to determine which factors have the most leverage on the design, and also, to find the optimum factor-level combination.

Table 5.17 The effects of GA parameters

Population Size	Mutation Rate	Parent Selection							
		Random Selection (1)				Roulette Wheel Selection (2)			
		Crossover Rules							
		1		2		1		2	
60	0.01	379732.6	408999.5	413107.6	439888.7	371137.0	419445.0	457153.8	383076.6
		485067.5	379732.6	539463.1	455848.8	533583.5	454778.6	550413.4	445936.1
	0.02	466064.0	422968.5	462874.6	402718.6	423396.1	393089.8	427269.6	450433.0
		500885.6	416850.4	534390.0	509856.0	531112.5	394193.5	505763.0	458513.8
	0.05	396230.0	419972.1	421883.4	427674.2	393685.5	374417.0	382942.4	357133.8
		470970.8	436524.8	502767.8	411562.6	501397.8	406547.2	518205.1	453646.5
	0.1	405285.8	372222.1	413849.1	387812.5	400936.7	386397.4	415742.2	411234.0
		460565.5	425425.0	465945.9	410945.1	484335.7	402618.3	489584.6	443367.8
120	0.01	434462.8	416213.4	430843.6	459693.9	406869.0	361728.8	400937.5	395017.2
		505114.0	384686.1	542916.9	418514.8	519504.9	415862.5	513274.2	417140.2
	0.02	390583.6	384388.7	378546.6	384232.6	385770.6	417240.5	414035.1	412978.3
		466234.4	374779.6	523261.0	421240.6	438223.3	452015.7	495320.0	394362.9
	0.05	397499.2	417673.5	380605.9	425293.6	409273.0	388268.8	425108.0	385574.9
		474603.1	409473.8	495152.8	449639.1	468996.5	389944.6	442840.7	404442.4
	0.1	451820.9	380595.2	397614.2	368271.7	398863.5	374794.7	398853.4	432511.9
		417216.0	382355.9	464544.6	446652.0	384595.1	364873.2	478000.1	389967.2
180	0.01	394624.8	411663.1	356185.3	415933.8	417545.5	408550.5	392904.8	405408.7
		417514.5	397020.2	470601.3	398322.2	466865.3	408951.9	510151.3	421475.1
	0.02	387439.2	357976.3	390964.9	396561.2	380555.9	349198.8	443186.4	395119.7
		473896.3	398146.0	511036.9	423426.2	462680.4	357546.1	492377.7	447589.8
	0.05	448359.0	380635.0	401937.8	393508.3	409600.4	399585.4	396523.9	377965.3
		446960.2	368041.3	489523.9	434287.9	469936.9	425339.5	455641.4	381719.2
	0.1	478197.8	382931.7	376292.8	408136.8	383565.1	369665.3	383744.4	396474.3
		434646.6	337234.4	494645.6	395740.4	404085.8	447605.8	432374.3	381287.9

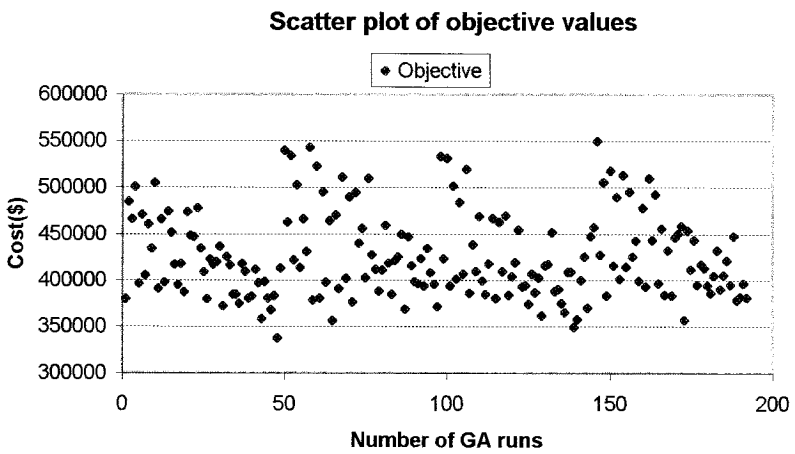


Figure 5.4 Scatter plot of the objective values

To compensate the random effect, four replication runs of the program are conducted for each factor-level combination. This leads to a total of 192 GA runs. Computation is stopped if the objective value does not improve after 100 continuous iterations in all cases. The computational results of objective values are summarized in Table 5.18. The maximum and minimum objective values achieved by the GA are 550413.4 and 337234.4 respectively. A scatter plot of the 192 objective values is given in Figure 5.4. About 80% of the objective values are within the range of 337234.4 to 550413.4, which means the variation is $\pm$10% of the average value of 425359 (The average of all the 192 numbers).

Table 5.18 The results of ANOVA

Source	Sum of Square	DOF	Mean Square	F ratio	P value	Significant?
Parent selection (PA)	5.376956E+08	1	5.376956E+08	0.246	0.62	No
Crossover Rule (CR)	1.25095E+10	1	1.25095E+10	5.73	0.018	Yes
Mutation rate (MU)	1.30665E+10	3	4.355499E+09	1.995	0.117	No
Population size (PO)	1.878288E+10	2	9.39144E+09	4.302	0.015	Yes
PA*CR	2.966792E+08	1	2.966792E+08	0.136	0.713	No
PA*MU	1.719011E+09	3	5.730035E+08	0.262	0.853	No
PA*PO	8.567164E+08	2	4.283582E+08	0.196	0.822	No
CR*MU	3.101199E+09	3	1.033733E+09	0.474	0.931	No
CR*PO	4.432816E+08	2	2.216408E+08	0.102	0.903	No
MU*PO	6.511823E+09	6	1.085304E+09	0.497	0.81	No
PA*CR*MU	2.245444E+09	3	7.484813E+08	0.343	0.794	No
PA*CR*PO	2188244	2	1094122	0.0005	1.000	No
PA*MU*PO	6.370951E+09	6	1.061825E+09	0.486	0.818	No
CR*MU*PO	3.831462E+09	6	6.38577E+08	0.293	0.94	No
PA*CR*MU*PO	3.863189E+09	6	6.438649E+08	0.295	0.938	No
Error	3.143442E+11	144	2.182946E+09			
Total Sum of Square	3.884827E+11	191				

The ANOVA results are shown in Table 5.19. In this table, the first column contains the source of variation. There are fifteen sources of variation: differences among groups (the number of levels for each source) and error. Here, "error" means unexplained variation. The second column lists the "Sum of Square" which is the sum of the squared

differences of each objective value from the mean of all objective values. The abbreviation "DOF" in the third column means degree of freedom and it is determined as follows:

1. The degree for groups is always equal to the number of groups minus one.
2. The degree for error is equal to the number of subjects minus the number of groups. Subjects refer to the number of data for each group.
3. The degree for total is the number of subjects minus one.

The fourth column contains the mean squares. Mean squares are estimates of variance and are computed by dividing the sum of squares by the degree of freedom. For example, the mean square for PO (9.39144E+09) is computed by dividing the sum of squares for PO (1.878288E+10) by the degree of freedom for PO (2). The fifth column contains the F ratio. It is computed by dividing the mean square for groups by the mean square for error. For example, the F ratio for PO is (9.39144E+09)/(2.182946E+09) = 4.302. The second last column is the probability value. It is the probability of obtaining an F greater than or equal to the one computed in the data.

The results indicate that of the four parameters, only the crossover rule and population size have significant effects (assuming a 5% level of confidence) on the improvement of objective values. There is no strong preference for parent selection method and any of the four mutation rates. The table also shows that the interactions between parameters have no strong impact on the performance of the GA.

Chapter 6

Conclusions and Future Work

6.1 Summary of This Study

The potential benefits of RMS can be achieved by reconfiguring its structure in such a way that each configuration corresponds to one product variant in the same family. The successful implementation of this strategy lies in efficient scheduling the production of a family of modular products and selecting the cell configurations. For this purpose, an integrated model has been developed. A genetic algorithm (GA) approach with parallel chromosome coding scheme is designed to simultaneously determine the processing sequence of modular products and select the system configuration for each product variant. The proposed GA differs from the conventional GA with serial chromosome coding in that each chromosome contains two sets of genes, one for job sequence decision and the other for cell configuration. The two aspects of the problem are handled simultaneously, which substantially improves the computing efficiency. The computational results indicate that the proposed method is very effective and can be used for solving relatively large problems.

6.2 Future Work

The following areas are worth further studying in the future:

(1) Integration of module selection, production scheduling and cell configuration decisions.

In this study, modular product scheduling and cell configuration selection in RMS are addressed. The structure of modular products is assumed known in advance. To reduce the overall cost of developing a family of modular products, it is necessary to integrate module selection, production scheduling and cell configuration decisions into a single model. Addressing one or two of them can only lead to sub-optimal solutions.

(2) Decision of multi-instances from the same module in a product variant

In this work, it is assumed that only one instance from each module is included in a product variant. In reality, a product variant usually contains multiple instances from the same module. A typical example is the computer memory cards. There are usually multiple memory cards in a computer. This type of problem will result in unequal number of jobs among cells. It would be a challenge to share the common GA program modules for looping operations for each cell while still maintaining the computational efficiency.

(3) System changeover time

The system changeover time is not considered in this study. Since the reconfigurable manufacturing system is mainly used for medium job size, the amount of system changeover time is relative small compared to the processing time of the whole batch of jobs. Hence, it is reasonable to ignore the setup time between jobs. If the lot size is further split into smaller ones, the system setup time becomes significant. This requires that the setup time be integrated into current model.

(4) Scheduling constraints

The unit machine idle cost is assumed the same for each machine type and all machines are assumed to be continuously running in this study. The product delivery time is assumed to be at the end of production planning horizon. These assumptions may not be always valid in real world. Such constraints need to be further explored.

(5) Product quality issues

Machining systems designed in different configurations have different quality performance. Due to the complexity of machining processes, it is difficult to predict the error propagation for machining systems and to analyze the impact of system configuration on quality variation.

References

1. Abdi, M.R., and Labib, A.W., 2003, A design strategy for reconfigurable manufacturing systems (RMSs) using analytical hierarchical process (AHP): a case study. *International Journal of Production Research*, Vol. 41, No.10, 2273 - 2299
2. Abdi, M.R., and Labib, A.W., 2004, Grouping and selecting products: the design key of Reconfigurable Manufacturing Systems (RMSs). *International Journal of Production Research*, Vol.42, No.3, 521-546
3. Benjaafar, S., 1995, Machine sharing in cellular manufacturing systems. In Kamrani, A.K., Parasei, H.R., and Liles, D. H., (eds). *Planning, Design and Analysis of Cellular Manufacturing Systems* (Elsevier Science)
4. Brandimarte, P., 1993, Routing and scheduling in a flexible job shop by taboo search. *Annals of Operation Research*, 41,157-183
5. Bruker, P., and Schlie, R., 1991, Job-shop scheduling with multi-purpose machines. *Computing*, 45, 369-375
6. Dauzere-Peres, S., and Paulli, J., 1997, An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem with tabu search. *Annals of Operations Research*, 70, 281-306
7. Deckro, R., Herbert, J., and Winkofsky, E., 1982, Multiple criteria job-shop scheduling. *Computer Operation Research*, 9 (4), 279-285
8. Flynn, B. B., and Jacobs, F. R., 1986, A simulation comparison of group technology with traditional job shop manufacturing. *International Journal of Production Research*, 24(5), 1171 - 1192
9. Fujita, K., Akagi, S., Yoneda, T. and Ishikawa, M., 1998, Simultaneous optimization of product family sharing system structure and configuration. *Proceedings of the 1998 ASME Design Engineering Technical Conference.* Paper No. DETC98/DFM-5722
10. Fujita, K., 1999, Product variety development and its optimization under modular architecture and module communalization. *Proceedings of the 1999 ASME Design Engineering Technical Conferences*, Las Vegas, 337-348

11. Fujita, K., 2002, Product variety optimization under modular architecture. *Computer-Aided Design*, Vol. 34, 953-965
12. Garey, M. R., Johnson, D. S., and Sethi, R., 1976, The complexity of folwshop and job-shop scheduling. *Mathematics of Operation Research*, 1, 117-129
13. Ghiassi, M., Devor, R., Dessouky, M., and Kijowski, B., 1984, An application of multiple criteria decision making principles for planning operations. *IEEE Transactions,* 16(2), 106-114
14. Giusti, F., Santochi, M., Dini, G., and Arioti, A., 1994, A reconfigurable assembly cell for mechanical products, *Annals of the CIRP,* Vol.43, No.1, 1-4
15. Grignon, P.M. and Fadel, G.M., 2004, A GA based configuration design optimization method. *Transactions of the ASME*, 6/Vol.126
16. Gonzalez,T., and Sahni, S.,1978, Flowshop and jobshop schedules: complexity and approximation. *Operations Research*, 26, 36-52
17. Gonzalez, J.P., and Otto, K.N., 2000, Modular platform-based product family design. *Proceedings of the DETC'00 ASME Design Engineering Technical Conferences and Computers and Information in Engineering conference*, Baltimore, 677-686
18. Gu, P., and Slevinsky, M., 2003, Mechanical bus for modular product design. *Annals of the CIRP,* 52-1, 113-116
19. He, D. W. and Kusiak, A., 1997, Design of assembly system for modular products. *IEEE Transactions on Robotics and Automation*, 13, 646-655
20. Hu, W., 1993, Modules for modular design of machine tools, *in Proceeding of International Conference on Engineering Design*, The Hague, 17-19 August, Heunsta Zurich, 1287-1294
21. Huang, G., Bin, S., and Halevi, G., 2003, Product platform identification and development for mass customization. *Annals of the CIRP*, 52-1, 117-120
22. Huang, G., and Kusiak, A., 1998, Modularity in design of products and systems. *IEEE Transaction on Systems, Man and Cybernetics-Part A: Systems and Humans*, Vol.28, No.1
23. Hurink,E., Jurisch, B., and Thole, M., 1994, Tabu search for the job shop scheduling problem with multi-purpose machines. *Operations Research Spektrum*, 15,205-215

24. Irani, S.A., Cavalier, T.M., and Cohen, P.H., 1993, Virtual manufacturing cells: exploiting layout design and intercell flows for the machine sharing problem. *International Journal of Production Research*, Vol.31, No.4, 791-810
25. Kacem, I., Hammadi, S., and Borne, P., 2002, Appoach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems. *IEEE Transactions on Systems, Man, and Cybertics, Part C*, 32(1), 1-13
26. Kacem, I., Hammadi, S., and Borne, P., 2002, Pareto-optimality approach for flexible job-shop scheduling problems: Hybridization of evolutionary algorithm and fuzzy logic. *Mathematics and Computers in Simulation*, 60, 245-276
27. Kamrani, A.K., and Gonzalez, R., 2003, A genetics algorithm-based solution methodology for modular design. *Journal of Intelligent Manufacturing*, 14, 599-616
28. Khoo, L.P., and Situmdrang, T.D., 2003, Solving the assembly configuration problem for modular products using an immune algorithm approach. *International Journal of Production Research*, Vol.41, No.15, 3419-3434
29. Kim, Y.K., Park, K., and Ko, J., 2003, A symbiotic evolutionary algorithm for the integration of process planning and job shop scheduling. *Computers and operations Research*, 30, 1151-1171
30. Kolonko, M., 1999, Some new results on simulated annealing applied to the job shop scheduling problem. *European Journal of Operational Research*, 113(1), 123-136
31. Kusiak, A., and Huang, C.C., 1996, Development of Modular products. *IEEE Transaction on Components, Packaging and Manufacturing Technology-Part A*, 19-4, 523-538
32. Lee, G.H., 1997, Reconfigurability consideration design of components and manufacturing systems. *Internatioal Journal of Advanced Manufacturing Technology*, Vol.13, 376-386
33. Leu, Y.Y, Matheson, L.A., and Rees, L.P., 1994, Assembly line balancing using genetic algorithms with heuristic-generated initial populations and multiple evaluation criteria. *Decision Science*, 25,4, pp.581
34. Mak, K.L., Lau, J.S.K., and Wang, X.X., 2005, A Genetic Scheduling Methodology for Virtual Cellular Manufacturing Systems: an Industrial Application. International *Journal of Production Research*, Vol.43, No.12, 2423-2450

35. Mastrolilli, M., and Gambardella, L.M., 2002, Effective neighborhood functions for the flexible job-shop problems. *Journal of Scheduling*, 3(1), 3-20

36. Mehrabi, M.G., Ulsoy, A.G. and Koren Y., 2000, Reconfigurable manufacturing systems: Key to future manufacturing. *Journal of Intelligent Manufacturing*, Vol.11, 403-419

37. Mehrabi, M.G., Ulsoy, A.G., Koren, Y. and Heytler P., 2002, Trends and perspectives in flexible and reconfigurable manufacturing systems. *Journal of Intelligent Manufacturing*, Vol.13, 135-146

38. Nayak, R.U., Chen, W., and Simpson, T.W., 2002, A variation-based method for product family design. *Engineering Optimization*, Vol. 34, No.1, 65-81

39. Norbis, M.L., and Smith J.M., 1988, A multiobjectie, multi-level heuristic for dynamic resource constrained scheduling problems. *European Journal of Operational Research*, 33, 30-41

40. Pirlot, M., 1996, General local search methods. *European Journal of Operational Research*, 92, 493-511

41. Rajendrana, C., and Ziegler, H., 2005, Two ant-colony algorithms for minimizing total flowtime in permutation flowshops. *Computers & Industrial Engineering*, 48, 789–797

42. Rekiek, B., Falkenauer, E., and Delchambre, A., 1997, Multi-Product Resource Planning. *Proceeding of the 1997 IEEE International Symposium on Assembly and Task Planning*, Marina del Rey, CA, 115-121

43. Rogers, G.G., 1990, Modular production systems: a control scheme for actuators, PhD thesis, Loughborough University, UK

44. Rogers, G.G., and Bottaci, L., 1997, Modular production systems: a new manufacturing paradigm. *Journal of Intelligent Manufacturing*, Vol.8, 147-156

45. Selvam, R.P., and Balasubramanian, K.N., 1985, Algorithmic grouping of operation sequences. *Engineering Costs and Production Economics*, 9, 125-134

46. Shanker, K., and Modi, B.K., 1999, A branch and bound based heuristic for multi-product resource constrained scheduling problem in FMS environment. *European Journal of Operational Research*, 113, 80-90

47. Simpson, J.A., Hocken, R.J., and Albus, J.S., 1982, The automated manufacturing research facility of the National Bureau of Standards. *Journal of Manufacturing Systems*, 1(1),17-32

48. Son, S.Y., Olsen, T.L., and Yip-Hoi, D., 2001, An approach to scalability and line balancing for reconfigurable manufacturing systems. Integrated Manufacturing Systems, 12/7, 500-511

49. Stafford, Jr.E.F., and Tseng, F.T., 2002, Two general models for a family of flowshop sequencing problems. *European Journal of Operational Research*, Vol.142, No.2, 282-293

50. Su, C.T., and Hsu, C.M., 1998, Multi-objective machine-part cell formation through parallel simulated annealing. *International Journal of Production Research*, Vol. 36, No. 8, 2185-2207

51. Travaini, E., Pedrazzoli, P., Rinaldi, R., and Boer, C.R., 2002, Methodological approach and reconfiguration tool for assembly systems. *Annals of the CIRP*, 51-1, 117-120

52. Tung, L.F., Li, L., and Nagi, R., 1999, Multi-objective scheduling for the hierarchical control of flexible manufacturing systems. *The International Journal of Flexible Manufacturing Systems*, 11, 379-409

53. Ulrich, K., and Tung, K., 1991, Fundamentals of product modularity. *In Issues in Design Manufacture/integration* (Sharon, A., Behun, R., Prinz, F., and Young L.), ASME, 73-79.

54. Van Wassenhove, L., and Gelders, L., 1980, Solving a bi-criterion scheduling problem. *European Journal of Operational Research*, 4, 42-48

55. Xia, W., and Wu, Z., 2005, An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems. *Computers &Industrial Engineering*, 48, 409-425

56. Xu, Z.H., and Liang, M., 2004, A cost model and genetic algorithm for the integration of module instance selection and assembly line design. M. E. project report, Department of Mechanical Engineering, University of Ottawa

57. Yang, T., and Peters, B.A., 1998, Flexible machine layout design for dynamic and uncertain production environments. *European Journal of Operational Research*, 108, 49-64

58. Yigit, A.S., Ulsoy, A.G., and Allahverdi, A., 2002, Optimizing modular product design for reconfigurable manufacturing. *Journal of Intelligent Manufacturing*, 13, 309-316

59. Yigit, A.S., and Allahverdi, A., 2003, Optimal selection of module instances for modular products in reconfirurable manufacturing systems. *International Journal of Production Research*, Vol.41, No.17, 4063-4074

60. Yeo, K.K., Young, J.K., and Yeo, K., 1996, Genetic algorithms for assembly line balancing with various objectives. *Computers Industrial Engineering*, Vol.30, No.3, 397-409

61. Yu, J., Yin, Y., Sheng, X., and Chen, Z., 2003, Modelling strategies for reconfigurable assembly systems. *Assembly Automation*, Vol.23, No.3, 266-272

62. Zhao, X.B., Wang, J.C. and Luo, Z.B., 2000, A stochastic model of a reconfigurable manufacturing system Part 1: A framework. *International Journal of Production Research*, Vol. 38, No. 10, 2273-2285

Appendix A

LINGO Program Code for Case Study

```
MODEL:
 !Solving the combined modular product scheduling and
 !production cell configuration selection problem;
 SETS:
  VAR/V1..V3/:DEMAND;
  !There are 3 product variants, each with a demand;
  POS/P1..P3/; ! There are 3 job positions to be filled in each cell;
  CON/F1..F3/; ! There are 3 possible configurations in each cell;
  CEL/C1..C4/; ! There are 4 reconfigurable manufacturing cells;
  CEL_VAR(CEL,VAR):Q,CT,DELT;
  !Q(C,J)is the job volume in each cell;
  !CT(C,J) is the job completion time in each cell;
  !DELT(C,J) is the completion time difference of product variant J;
  VAR_POS(CEL,VAR,POS): X,CTP;
  !X(C,J,P)is the job-position assignment variable,
  !X(C,J,P)=1 if job J is assigned
  !to sequence position P in cell C, otherwise, 0;
  !CTP(C,J,P)is the completion time of job J
  !if it is assigned to position P in cell C;
  VAR_POS_VAR(VAR_POS,VAR)|&4 #NE# &2 #AND# &3 #LT# @SIZE(POS): Y;
  !Y(C,J,P,K)=1 if job J is assigned to position P and immediately
 !followed by job K(K<>J)in cell c, otherwise, 0;
  VAR_CON(CEL,VAR,CON): Z,FEA,UMH,UMI,UPT,CTF;
  !Z(C,J,F)= 1 if job J uses configuration F in cell C, otherwise,0
  !FEA(C,J,F)=1 if configuration F is feasible for job J in cell C,
 otherwise,0
  !UMH(C,J,F) is the number of machine distance that a component has to
 travel if configuration F is used to process job J in cell C;
  !UMI(C,J,F) is the number of idle running machine
  if configuration F is used to process job J in cell C;
  !UPT(C,J,F) is the unit processing time of job J
  if configuration F is used to process job J in cell C;
  !CTF(C,J,F) is the completion time of job J
  if configuration F is used to process job J in cell C;
  VAR_VAR_COST(CEL,VAR,CON,VAR,CON)|&4 #NE# &2: JFKF;
  !JFKF(C,J,FJ,K,FK)is the reconfiguration cost of changing system from
 job J to K(K<>J) here, job J uses configuarion FJ and job K uses
 configuration FK;
  CON_CON(CEL,CON,CON):FFC;
  !FFC(C,F1,F2) is the reconfiguration cost from cell configuration F1
 to F2 in cell C;
 ENDSETS

  ! Model;
  ! Calculating the job quantities in each cell;
  @FOR( CEL(C):
    @FOR( VAR(J):
     Q(C,J)= DEMAND(J)
    )
  );
```

```
  ! Calculating the job-job configuratin cost based on
    the configuation-configuration cost in each cell;
  @FOR( CEL(C):
    @FOR( VAR(J):
      @FOR( CON(FJ):
        @FOR( VAR(K)|K #NE# J:
          @FOR( CON(FK):
            JFKF(C,J,FJ,K,FK)= FFC(C,FJ,FK)
          )
        )
      )
    )
  );

  ! Calculating job processing time based on the configuration
selected;
  @FOR( CEL(C):
    @FOR( VAR(J):
      @FOR( VAR(F):
        CTF(C,J,F)= Q(C,J)* UPT(C,J,F)
      )
    )
  );

  ! Completion time of the first position job in the sequence;
  @FOR( CEL(C):
    @FOR( VAR(J):
      @FOR( POS(P)|P #EQ# 1:
        @SUM( CON(F):
          Z(C,J,F)* CTF(C,J,F))= CTP(C,J,P)
      )
    )
  );

  ! Completion time of job that is not in the first position in the
   sequence;
  @FOR( CEL(C):
    @FOR( VAR(J):
      @FOR( POS(P)|P #GT# 1:
        @SUM( CON(FJ):
          Z(C,J,FJ)*CTF(C,J,FJ))+
           @SUM( VAR(I)|I #NE# J:
             @SUM( POS(E)|E #LT# P:
               @SUM( CON(FI):
                 X(C,I,E)*Z(C,I,FI)* CTF(C,I,FI))))= CTP(C,J,P)
      )
    )
  );

  ! Calculating the completion time of all batches
    of module instances for a product variant;
  @FOR( CEL(C):
    @FOR( VAR(J):
      @SUM( POS(P):
        X(C,J,P)* CTP(C,J,P))=CT(C,J)
    )
```

```
 );

 ! Calculating the completion time difference of all batches
   of module instances for a product variant;
 @FOR( CEL(C):
   @FOR( VAR(J):
   DELT(C,J)=@SMAX(CT(1,J),CT(2,J),CT(3,J),CT(4,J),CT(5,J))-CT(C,J)
   )
 );

! Objective;
MIN =
 @SUM( CEL(C):
   @SUM( POS(P)|P #LT# @SIZE(POS):
     @SUM( VAR(J):
       @SUM( VAR(K)|K #NE# J:
         @SUM( CON(FJ):
           @SUM( CON(FK):
             Y(C,J,P,K)*Z(C,J,FJ)*Z(C,K,FK)*JFKF(C,J,FJ,K,FK)))))))
+ @SUM( CEL(C):
    @SUM( VAR(J):
      @SUM( CON(F):
        Z(C,J,F)* UM * Q(C,J)* UMH(C,J,F))))
+ @SUM( CEL(C):
    @SUM( VAR(J):
      @SUM( CON(F):
        Z(C,J,F)* UI * Q(C,J)* UMI(C,J,F)* UPT(C,J,F))))
+ @SUM( CEL(C):
    @SUM( VAR(J):
      UW* DELT(C,J)));

 ! Each position  is filled by only one job in a cell;
 @FOR( CEL(C):
   @FOR( POS(P):
     @SUM( VAR(J):
       X(C,J,P))=1
   )
 );
 ! Each job is assigned to only one sequence position in a cell;
 @FOR( CEL(C):
   @FOR( VAR(J):
     @SUM( POS(P):
       X(C,J,P))=1
   )
 );

! Each job is assigned to one and only one configuration in a cell;
 @FOR( CEL(C):
   @FOR( VAR(J):
     @SUM( CON(F):
       Z(C,J,F))=1
   )
 );

 ! Both configurations should be feasible if they are selected to
   process two consecutive jobs which are arranged one after another
```

```
  in the job sequence in a cell;
@FOR( CEL(C):
  @FOR( VAR(J):
    @FOR( VAR(K)|K #NE# J:
      @SUM( VAR_CON(C,J,FJ)|FEA(C,J,FJ) #EQ# 1:
        @SUM( VAR_CON(C,K,FK)|FEA(C,K,FK) #EQ# 1:
          Z(C,J,FJ)*Z(C,K,FK)))=1
    )
  )
);

! Job J will be assigned to sequence position P
  if job J precedes the job in sequence position P+1;
@FOR( CEL(C):
  @FOR( VAR(J):
    @FOR( POS(P)|P #LT# @SIZE(POS):
      @SUM( VAR(K)| K #NE# J:
        Y(C,J,P,K))=X(C,J,P)
    )
  )
 );

! Job J will be assigned to sequence position P
  if job J follows the job in sequence position P-1;
@FOR( CEL(C):
  @FOR( VAR(J):
    @FOR( POS(P)|P #GT# 1:
      @SUM( VAR(K)| K #NE# J:
        Y(C,K,P-1,J))=X(C,J,P)
    )
  )
 );

! Make the X's,Y's,Z's binary;
@FOR( VAR_POS: @BIN( X));
@FOR( VAR_POS_VAR: @BIN( Y));
@FOR( VAR_CON: @BIN( Z));

DATA:
 UM = 2E-2;
 UI = 3E-3;
 UW = 5E-2;

 DEMAND =1265,3135,1010;

 UMH = 8,1000,1000
       8,6,5
       8,6,1000

       12,1000,1000
       12,7,4
       12,7,1000

       6,1000,1000
       1000,6,1000
       1000,1000,6
```

```
       7,1000,1000
       7,4,1000
       7,1000,1000;

UMI = 0,1000,1000
       3,1,0
       2,0,1000

       0,1000,1000
       8,3,0
       5,0,1000

       0,1000,1000
       1000,0,1000
       1000,1000,0

       0,1000,1000
       3,0,1000
       0,1000,1000;

FEA = 1,0,0
       1,1,1
       1,1,0

       1,0,0
       1,1,1
       1,1,0

       1,0,0
       0,1,0
       0,0,1

       1,0,0
       1,1,0
       1,0,0;

UPT = 465,10000,10000
       380,380,380
       415,415,10000

       250,10000,10000
       119,119,119
       170,170,10000

       308,10000,10000
       10000,312,10000
       10000,10000,310

       298,10000,10000
       208,208,10000
       298,10000,10000;

FFC = 0,1926,1782
       2154,0,1536
       2118,1644,0
```

```
        0,1744,1159
        2716,0,158
        2558,632,0

        0,1500,1500
        1500,0,1500
        1500,1500,0

        0,256,10000
        1024,0,10000
        10000,10000,0;

 ENDDATA
END
```

Appendix B

Visual Basic 6.0 Code for GA with Parallel Chromosome Coding

```
'Progam Name: Solving the Combined Modular Product Scheduling and Production Cell
'             Reconfiguration Problem: a GA Approach with Parallel Chromosome
'             Coding
'Programmer:  Hegui Ye
'Date:        March 08,2005
'Decription:  This project aims at finding the minimum production cost of
'             producing a family of modular products,provided the structure of
'             modular products and the manufacturing systems are known.
'             This project incorpoates the following production cost elements:
'             unit processing time, material handling cost, machine relocation cost,
'             machine idle time costs, and work-in-process holding costs.
'Folder:      "C:\Documents and Settings\HEGUI YE\My Documents\GA\

Option Explicit

Public Const sngUM = 0.02
Public Const sngUI = 0.003
Public Const sngUW = 0.05
Public Const BigM = 1000

'input and output variables
Public TC As Integer, TJ As Integer, MTI As Integer, TF() As Integer
Public Q() As Integer, FEA() As Integer, FFC() As Integer
Public sngUMH() As Integer, sngUMI() As Integer, sngUPT() As Single
Public X() As Integer, Z() As Integer

'Results-form related variables
Public lngMaxRunTime As Long, lngElapseTime As Long, lngElapseTime1 As Long
Public lngMaxIter As Long, lngNumIter As Long, lngStartTime As Long
Public lngStartTime1 As Long, sngMutRate As Single
Public ParentChoice As Integer, CrossRule As Integer, blnFlag As Boolean

'Chromosome related variables
Public Parent() As Integer, OldGenVal() As Integer
Public PopSize As Integer, Gene() As Integer, sngFitness() As Single
Public BuffGen As Integer, MutNum As Integer, NewP As Integer
Public sngFitNew As Single, sngBuffFit As Single
Public MaxRunTimeFlag As Boolean, MaxIterationFlag As Boolean
Public TerminalCondition As Boolean

'Index variables: c--cell, j--job, p--population, pp--job position, f--configuration
```

```
'          i--collection item only, k--offspring index, a and b--temporary index
Public a As Integer, b As Integer, c As Integer, d As Integer, e As Integer
Public f As Integer, i As Integer, j As Integer, k As Integer, p As Integer
Public pp As Integer, pf As Integer, n As Integer
```

```
Sub GA()

Dim lngTotalElapseTime As Long
Dim clnTempSet As New Collection, clnFinalSet As New Collection
Dim sngFitCriteriion As Single, sngFitCriterion() As Single, sngInitFitness As Single
Dim TerminalCounter As Integer, AccumulateCounter As Integer

ReDim OldGenVal(TC * TJ, 2), sngFitCriterion(105)
ReDim sngFitness(3 * PopSize), Gene(3 * PopSize, TC * TJ, 2)
ReDim X(TC, TJ, TJ), Z(TC, TJ, MTI)

Open "C:\Documents and Settings\HEGUI YE\My Documents\GA\Data3-temp.txt" For
Output As #7
Open "C:\Documents and Settings\HEGUI YE\My Documents\GA\Data3-Time.txt" For
Output As #8
Open "C:\Documents and Settings\HEGUI YE\My Documents\GA\Data3-Fitness.txt"
For Output As #9

'Initialization
lngStartTime1 = 0
lngElapseTime1 = 0
lngNumIter = 0
TerminalCounter = 0
AccumulateCounter = 0
sngFitCriterion(0) = 0
TerminalCondition = False

'set timer
lngStartTime1 = Hour(Time) * 3600 + Minute(Time) * 60 + Second(Time)

'Generate initial population
'genes representing job-position assignments
Set clnTempSet = Nothing
Set clnFinalSet = Nothing
For p = 1 To PopSize
  For c = 0 To TC - 1
    k = c * TJ
    For j = 1 To TJ
      a = k + j
      Do
```

```
      Gene(p, a, 1) = Int(TJ * Rnd()) + 1
      If j = 1 Then
       clnFinalSet.Add Item:=Gene(p, a, 1)
       Exit Do
      End If
      If j >= 2 Then
       For i = 1 To clnFinalSet.Count
        If Gene(p, a, 1) <> clnFinalSet.Item(i) Then
         clnTempSet.Add Item:=Gene(p, a, 1)
        End If
       Next i
       If clnTempSet.Count = clnFinalSet.Count Then
        clnFinalSet.Add Item:=Gene(p, a, 1)
       End If
       Set clnTempSet = Nothing
      End If
     Loop Until clnFinalSet.Count = j
    Next j
    For j = 1 To TJ
     b = k + j
     'genes representing job-position assignment
     Gene(p, b, 1) = clnFinalSet.Item(j)
     'genes representing job-configuration selection
     Do
      Gene(p, b, 2) = Int(TF(c + 1) * Rnd()) + 1
     Loop Until FEA(c + 1, j, Gene(p, b, 2)) = 1
    Next j
    Set clnFinalSet = Nothing
   Next c
   'calculating objective function value and assign it as fitness value
   sngFitness(p) = sngObjective(p)
  Next p
   For p = 2 To PopSize
    If sngFitness(p) < sngFitness(1) Then
     sngBuffFit = sngFitness(p)
     sngFitness(p) = sngFitness(1)
     sngFitness(1) = sngBuffFit
     For c = 0 To TC - 1
      For j = 1 To TJ
       a = c * TJ + j
       For pf = 1 To 2
       BuffGen = Gene(p, a, pf)
       Gene(p, a, pf) = Gene(1, a, pf)
       Gene(1, a, pf) = BuffGen
       Next pf
      Next j
```

```
    Next c
   End If
  Next p
sngInitFitness = sngFitness(1)
'iterate until meet the stopping criterion
'i.e. lngNumIter = lngMaxIter(maximum iteration number)
'or Elapsed run time = Maximum run time
Write #8, "Elapsed Run Time for each iteration= "
Write #9, "Objective = "
Do
  d = 0
  e = 0
  'generate offsprings until the number of offsprings =2 Popsize
  Do
    Call Select_Parents
    Call crossover
    d = d + 1
    e = e + 1
    sngFitness(PopSize + d) = sngObjective(PopSize + d)
    sngFitness(PopSize * 2 + e) = sngObjective(PopSize * 2 + e)
  Loop While d + e < 2 * PopSize

  'randomly select MutNuM chromosomes for mutation
  MutNum = Int(sngMutRate * (PopSize * (TC + TJ) * Rnd())) + 1
  For p = 1 To MutNum
    'randomly selecting one chromosome
    NewP = Int(PopSize * Rnd()) + 1

    'call sub mutation
    Call Mutation

    'Calculate the fitness value after mutation
    sngFitness(NewP) = sngObjective(NewP)
    If sngFitness(NewP) > sngFitNew Then
      sngFitness(NewP) = sngFitNew
      For c = 0 To TC - 1
        For j = 1 To TJ
          a = c * TJ + j
          For pf = 1 To 2
          Gene(NewP, a, pf) = OldGenVal(a, pf)
          Next pf
        Next j
      Next c
    End If
  Next p
```

```
For n = 1 To 2
 For p = 1 To PopSize
  For i = 1 To d
  k = PopSize * n + i
  If sngFitness(k) < sngFitness(p) Then
   sngBuffFit = sngFitness(k)
   sngFitness(k) = sngFitness(p)
   sngFitness(p) = sngBuffFit

   'if offspring chromosome survives,the following code segments
   'exchange all genes between offsprings and their parents
   For c = 0 To TC - 1
    For j = 1 To TJ
     a = c * TJ + j
     For pf = 1 To 2
     BuffGen = Gene(k, a, pf)
     Gene(k, a, pf) = Gene(p, a, pf)
     Gene(p, a, pf) = BuffGen
     Next pf
    Next j
   Next c
  End If
  Next i
 Next p
Next n

'compare the fitness values of current populations,
'the smallest one is retained in chromosome number one
For p = 2 To PopSize
 If sngFitness(p) < sngFitness(1) Then
  sngBuffFit = sngFitness(p)
  sngFitness(p) = sngFitness(1)
  sngFitness(1) = sngBuffFit
  For c = 0 To TC - 1
   For j = 1 To TJ
    a = c * TJ + j
    For pf = 1 To 2
    BuffGen = Gene(p, a, pf)
    Gene(p, a, pf) = Gene(1, a, pf)
    Gene(1, a, pf) = BuffGen
    Next pf
   Next j
  Next c
 End If
Next p
```

```
  'if the fitness value doesn't improve after 20 iterations,
  'the programm will be ended
   If sngFitness(1) = sngInitFitness Then
     TerminalCounter = TerminalCounter + 1
     sngFitCriterion(TerminalCounter) = sngFitness(1)
     If sngFitCriterion(TerminalCounter) = sngFitCriterion(TerminalCounter - 1) Then
       AccumulateCounter = AccumulateCounter + 1
       If AccumulateCounter = 100 Then
         TerminalCondition = True
       End If
     Else
       AccumulateCounter = 0
     End If
   Else
     For i = 1 To TerminalCounter
       sngFitCriterion(TerminalCounter) = 0
     Next i
     TerminalCounter = 0
   End If
   sngInitFitness = sngFitness(1)
  'Display elapsed time GA has used
  lngElapseTime1 = Hour(Time) * 3600 + Minute(Time) * 60 + Second(Time) _
- lngStartTime1
  Write #7, lngNumIter
  Write #8, lngElapseTime1
  Write #9, sngFitness(1)

  lngNumIter = lngNumIter + 1
  If (lngNumIter Mod 5) = 0 Then
    frmGAInformation.lblStatus.Caption = "Calculating Iteration No..." & lngNumIter
    frmGAInformation.lblStatus.Refresh
  End If
  'decide stopping criterion
  If MaxRunTimeFlag = True And lngElapseTime1 >= lngMaxRunTime Then
    TerminalCondition = True
  End If
  If MaxIterationFlag = True And lngNumIter >= lngMaxIter Then
    TerminalCondition = True
  End If
Loop Until TerminalCondition = True

'find out the X,Y values for the best chromosome no one
For c = 0 To TC - 1
  For j = 1 To TJ
    a = c * TJ + j
    For pp = 1 To TJ
```

```
      If Gene(1, a, 1) = pp Then
        X(c + 1, j, pp) = 1
      Else
        X(c + 1, j, pp) = 0
      End If
    Next pp
    For f = 1 To TF(c + 1)
      If Gene(1, a, 2) = f Then
        Z(c + 1, j, f) = 1
      Else
        Z(c + 1, j, f) = 0
      End If
    Next f
  Next j
Next c
Close #7
Close #8
Close #9
End Sub
```

```
Sub Select_Parents()

Dim sngSumFit As Single, sngPointer1 As Single, sngPointer2 As Single
Dim sngRouWheel() As Single
ReDim sngRouWheel(PopSize), Parent(2)

'Random parent selection
If ParentChoice = 1 Then
  Do
    Parent(1) = Fix(PopSize * Rnd())
  Loop Until Parent(1) <> 0
  Do
    Parent(2) = Fix(PopSize * Rnd())
  Loop While (Parent(2) = Parent(1) Or Parent(2) = 0 Or sngFitness(Parent(2)) = 0)
End If

'Roullete wheel parent selection
If ParentChoice = 2 Then
  sngSumFit = 0
  sngRouWheel(0) = 0
  For p = 1 To PopSize
    sngSumFit = sngSumFit + 1 / sngFitness(p)
  Next p
  For p = 1 To PopSize
    sngRouWheel(p) = sngRouWheel(p - 1) + 1 / (sngFitness(p) * sngSumFit)
```

```
  Next p
  sngPointer1 = Rnd()
  For p = 1 To PopSize
   If sngRouWheel(p) > sngPointer1 Then Exit For
  Next p
  Parent(1) = p
  Do
   sngPointer2 = Rnd()
   For p = 1 To PopSize
    If sngRouWheel(p) > sngPointer2 Then Exit For
   Next p
   Parent(2) = p
  Loop While Parent(2) = Parent(1)
End If
End Sub
```

```
Sub crossover()

Dim Cut1  As Integer, Cut2 As Integer, BuffCut As Integer, dd As Integer
Dim ee As Integer
Dim clnParOneSet As New Collection, clnParTwoSet As New Collection

dd = PopSize + d + 1
ee = PopSize * 2 + e + 1
Select Case CrossRule
Case 1  'one-point crossover
 'for small-size problem, one-point cut crossover is adopted.
 'generate cut point for genes representing job-position assignments
 For c = 0 To TC - 1
  k = c * TJ
  Cut1 = Fix(TJ * Rnd())
  If Cut1 = 0 Then
    Cut1 = 1
  End If
  'retain genes before the one-point cut
  For j = 1 To Cut1
   a = k + j
   'the first offspring retains genes from parent1 before the cut-point
   Gene(dd, a, 1) = Gene(Parent(1), a, 1)
   Gene(dd, a, 2) = Gene(Parent(1), a, 2)
   'the second offspring retains genes from parent2 before the cut-point
   Gene(ee, a, 1) = Gene(Parent(2), a, 1)
   Gene(ee, a, 2) = Gene(Parent(2), a, 2)
  Next j

  'Offsprings' genes after the cut1 will follow the sequence of their
```

```
'counterpart parents,i.e. genes after the cut1 in offspring1 follow
'the sequence of genes in parent2 and genes after the cut1 in offspring2
'follow the sequence of genes in parent1
Set clnParOneSet = Nothing
Set clnParTwoSet = Nothing
For i = 1 To TJ
 'Compare each gene in parent2 with genes after cut point in parent1,
 'if same,add those genes in parent2 that match genes after cut point
 'in parent1 to clnParOne.so collection "clnParOne" includes all the
 'genes after cut point in parent1,but these genes have been reordered
 'according to the gene sequence in parent2
 For j = Cut1 + 1 To TJ
  If Gene(Parent(2), k + i, 1) = Gene(Parent(1), k + j, 1) Then
   clnParOneSet.Add Item:=Gene(Parent(2), k + i, 1)
   Exit For
  End If
 Next j
 'Compare each gene in parent1 with genes after cut point in parent2,
 'if same,add those genes in parent1 that match the genes
 'after cut point in parent1 to clnParTwo.so collection "clnParTwo"
 'includes all the genes after cut point in parent2,but these genes
 'have been reordered according to the gene sequence in parent1
 For j = Cut1 + 1 To TJ
  If Gene(Parent(1), k + i, 1) = Gene(Parent(2), k + j, 1) Then
   clnParTwoSet.Add Item:=Gene(Parent(1), k + i, 1)
   Exit For
  End If
 Next j
Next i

For j = Cut1 + 1 To TJ
 a = k + j
 'restore value of genes representing job-position assignments
 Gene(dd, a, 1) = clnParOneSet.Item(j - Cut1)
 Gene(ee, a, 1) = clnParTwoSet.Item(j - Cut1)
 'for genes representing job-configuration selection
 'the first offspring gains genes from parent2 after the cut-point
 Gene(dd, a, 2) = Gene(Parent(2), a, 2)
 'the second offspring gains genes from parent1 after the cut-point
 Gene(ee, a, 2) = Gene(Parent(1), a, 2)
Next j
Set clnParOneSet = Nothing
Set clnParTwoSet = Nothing
Next c

Case 2  'two-points cutting
```

```
'generate cut point for genes representing job-position assignments
For c = 0 To TC - 1
  k = c * TJ
  Cut1 = Fix(TJ * Rnd())
  If Cut1 = 0 Then
    Cut1 = 1
  End If
  Do
    Cut2 = Fix(TJ * Rnd())
    If Cut2 = 0 Then
      Cut2 = TJ - 1
    End If
  Loop While Cut2 = Cut1
  'exchange the value of Cut2 and Cut1 if Cut2 < Cut1
  If Cut2 < Cut1 Then
    BuffCut = Cut1
    Cut1 = Cut2
    Cut2 = BuffCut
  End If
  'retain genes before the first cut point
  For j = 1 To Cut1
    a = k + j
    'the first offspring retains genes from parent1,the second offspring
    'retains genes from parent2 before the first cut-point
    'for genes representing job-position assignments
    Gene(dd, a, 1) = Gene(Parent(1), a, 1)
    Gene(ee, a, 1) = Gene(Parent(2), a, 1)
    'for genes representing job-configuration selection
    Gene(dd, a, 2) = Gene(Parent(1), a, 2)
    Gene(ee, a, 2) = Gene(Parent(2), a, 2)
  Next j
  'retain genes after the second cut point
  For j = Cut2 To TJ
    a = k + j
    'the first offspring retains genes from parent1, the second offspring
    'retains genes from parent2 after the second cut-point
    'for genes representing job-position assignments
    Gene(dd, a, 1) = Gene(Parent(1), a, 1)
    Gene(ee, a, 1) = Gene(Parent(2), a, 1)
    'for genes representing job-configuration selection
    Gene(dd, a, 2) = Gene(Parent(1), a, 2)
    Gene(ee, a, 2) = Gene(Parent(2), a, 2)
  Next j

  'Offsprings' genes between the two cut-points will follow the sequence
  'of their counterpart parents,i.e. genes between the two cut-points
```

```
'in offspring1 follow the sequence of genes in parent2
'and genes between the two cut-points in offspring2 follow
'the sequence of genes in parent1
If Cut1 + 1 <= Cut2 - 1 Then
  Set clnParOneSet = Nothing
  Set clnParTwoSet = Nothing
  For i = 1 To TJ
    'Compare each gene in parent2 with genes between the two cut-points
    'in parent1,if same,add those genes in parent2 that match genes
    'between the two cut-points in parent1 to clnParOne.so collection
    '"clnParOne" includes all the genes between the two cut-points
    'in parent1,but these genes have been reordered according to
    'the gene sequence in parent2
    For j = Cut1 + 1 To Cut2 - 1
      If Gene(Parent(2), k + i, 1) = Gene(Parent(1), k + j, 1) Then
        clnParOneSet.Add Item:=Gene(Parent(2), k + i, 1)
        Exit For
      End If
    Next j
    'Compare each gene in parent1 with genes between the two cut-points
    'in parent2,if same,add those genes in parent1 that match the genes
    'between the two cut-points in parent1 to clnParTwo.so collection
    '"clnParTwo" includes all the genes between the two cut-points
    'in parent2,but these genes have been reordered according to
    'the gene sequence in parent1
    For j = Cut1 + 1 To Cut2 - 1
      If Gene(Parent(1), k + i, 1) = Gene(Parent(2), k + j, 1) Then
        clnParTwoSet.Add Item:=Gene(Parent(1), k + i, 1)
        Exit For
      End If
    Next j
  Next i

  For j = Cut1 + 1 To Cut2 - 1
    a = k + j
    'restore value of genes representing job-position assignments
    Gene(dd, a, 1) = clnParOneSet.Item(j - Cut1)
    Gene(ee, a, 1) = clnParTwoSet.Item(j - Cut1)
    'for genes representing job-configuration selection
    Gene(dd, a, 2) = Gene(Parent(2), a, 2)
    Gene(ee, a, 2) = Gene(Parent(1), a, 2)
  Next j
 End If
Next c
End Select
```

```
End Sub
```

```
Sub Mutation()

Dim MutPos1 As Integer

  'keep the fitness value before mutation
  sngFitNew = sngFitness(NewP)
  'keep the genes's values before mutation
  For c = 0 To TC - 1
    For j = 1 To TJ
      a = c * TJ + j
      For pf = 1 To 2
      OldGenVal(a, pf) = Gene(NewP, a, pf)
      Next pf
    Next j
  Next c
  'randomly select a cell to mutate
  c = Int(TC * Rnd())
  k = c * TJ
  'randomly select one mutation position for genes representing
  'job-position assignment
  MutPos1 = Int(TJ * Rnd()) + 1
  a = k + MutPos1
  Do
    'radomly generate a value for the newly selected gene,
    'it should not be the same as oldGenVal(a,1)
    Gene(NewP, a, 1) = Int(TJ * Rnd()) + 1
  Loop Until Gene(NewP, a, 1) <> OldGenVal(a, 1)
  Select Case MutPos1
  Case 1
    For i = 2 To TJ
      If Gene(NewP, a, 1) = OldGenVal(k + i, 1) Then
        Gene(NewP, k + i, 1) = OldGenVal(a, 1)
        Gene(NewP, a, 1) = OldGenVal(k + i, 1)
        Exit For
      End If
    Next i
  Case TJ
    For i = MutPos1 - 1 To 1 Step -1
      If Gene(NewP, a, 1) = OldGenVal(k + i, 1) Then
        Gene(NewP, k + i, 1) = OldGenVal(a, 1)
        Gene(NewP, a, 1) = OldGenVal(k + i, 1)
        Exit For
      End If
    Next i
```

```
    Case Else
      'Search backwards to find a gene with the new generated value
      'swift the gene value if found,otherwise search forwards to
      'find one to match that new value
      For i = MutPos1 - 1 To 1 Step -1
        If Gene(NewP, a, 1) = OldGenVal(k + i, 1) Then
          Gene(NewP, k + i, 1) = OldGenVal(a, 1)
          Gene(NewP, a, 1) = OldGenVal(k + i, 1)
          Exit For
        End If
      Next i
      'search forwards
      For i = MutPos1 + 1 To TJ
        If Gene(NewP, a, 1) = OldGenVal(k + i, 1) Then
          Gene(NewP, k + i, 1) = OldGenVal(a, 1)
          Gene(NewP, a, 1) = OldGenVal(k + i, 1)
          Exit For
        End If
      Next i
    End Select
    'radomly genrate a value for the selected gene,
    'subject to the feasibility constraints
    Do
      Gene(NewP, a, 2) = Int(TF(c + 1) * Rnd()) + 1
    Loop Until FEA(c + 1, MutPos1, Gene(NewP, a, 2)) = 1

End Sub
```

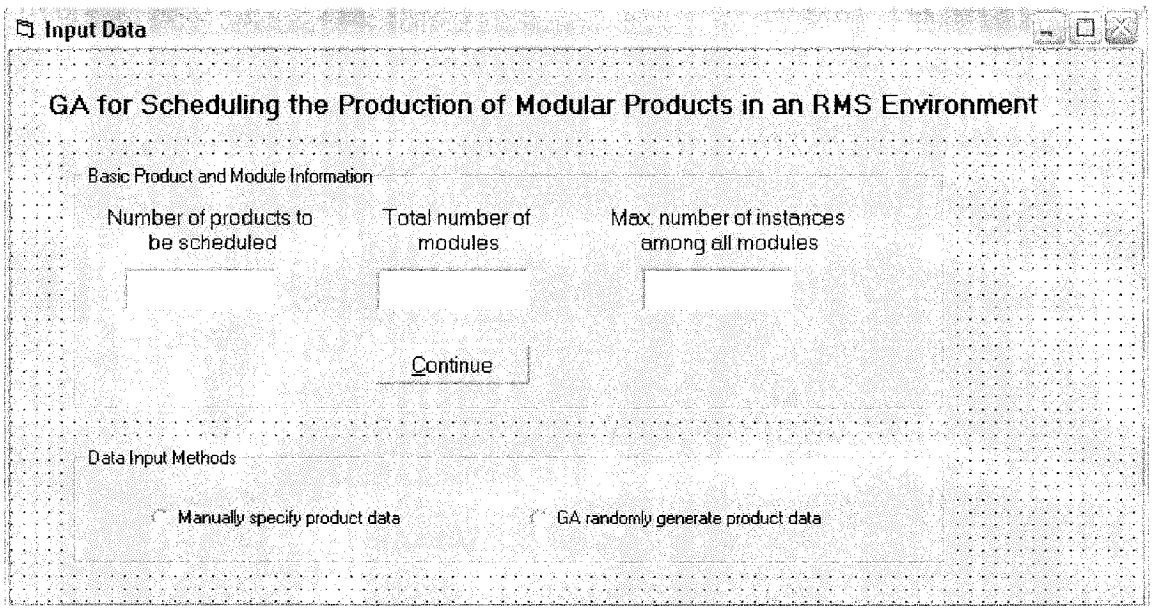
```
Function sngObjective(kk As Integer) As Single

Dim GenPos As Integer, GenPosOne As Integer, GenPosTwo As Integer
Dim FirstConf As Integer, SecondConf As Integer, g As Integer
Dim sngMIC As Single, sngMHC As Single, sngWIP As Single
Dim sngCFC As Single, sngCompTime() As Single, sngBuffTime() As Single
Dim sngTempTimeSet() As Single
Dim sngTotalDelt As Single, sngPrecedeJobs As Single, sngProcessTime() As Single
Dim clnFinGenPos As New Collection, clnFinGenCon As New Collection

ReDim sngCompTime(TC, TJ), sngBuffTime(TC, TJ)
ReDim sngDeltTime(TC, TJ), sngProcessTime(TC, TJ, TJ)

sngMIC = 0
sngMHC = 0
sngCFC = 0
sngWIP = 0
```

```
sngTotalDelt = 0

For c = 1 To TC
 a = (c - 1) * TJ
 For j = 1 To TJ
  b = a + j
  'Idle time cost of machine tools
  sngMIC = sngMIC + sngUI * Q(c, j) * sngUMI(c, j, Gene(kk, b, 2)) _
  * sngUPT(c, j, Gene(kk, b, 2))
  'Material handling cost
  sngMHC = sngMHC + sngUM * Q(c, j) * sngUMH(c, j, Gene(kk, b, 2))
 Next j

 'System reconfiguration costs
 For GenPosOne = 1 To TJ - 1
  For j = 1 To TJ
   b = a + j
   If Gene(kk, b, 1) = GenPosOne Then
    FirstConf = Gene(kk, b, 2)
    Exit For
   End If
  Next j
  GenPosTwo = GenPosOne + 1
  For j = 1 To TJ
   b = a + j
   If Gene(kk, b, 1) = GenPosTwo Then
    SecondConf = Gene(kk, b, 2)
    Exit For
   End If
  Next j
  sngCFC = sngCFC + FFC(c, FirstConf, SecondConf)
 Next GenPosOne

 'WIP holding cost
 'calculate the processing time of each job in each cell
 For GenPos = 1 To TJ
  For j = 1 To TJ
   b = a + j
   If Gene(kk, b, 1) = GenPos Then
    sngProcessTime(c, j, GenPos) = Q(c, j) * sngUPT(c, j, Gene(kk, b, 2))
    Exit For
   End If
  Next j
 Next GenPos
 'if a job is assigned in the first position in the sequence
 'the completion time of that job is the processing time of that job
```

```
  For j = 1 To TJ
   If Gene(kk, a + j, 1) = 1 Then
     sngCompTime(c, j) = sngProcessTime(c, j, 1)
     Exit For
   End If
  Next j

  'the completion time of jobs that are not in the first position in the sequence
  For GenPos = 2 To TJ
   For j = 1 To TJ
    If Gene(kk, a + j, 1) = GenPos Then
     'calculate the completion time of jobs that precedes job in position "Genpos".
     sngPrecedeJobs = 0
     For g = 1 To TJ
      For i = 1 To GenPos - 1
       sngPrecedeJobs = sngPrecedeJobs + sngProcessTime(c, g, i)
      Next i
     Next g
     'calculate the completion time of job in position "Genpos"
     sngCompTime(c, j) = sngPrecedeJobs + sngProcessTime(c, j, GenPos)
     Exit For
    End If
   Next j
  Next GenPos
  'buffer the completion time
  For j = 1 To TJ
   sngBuffTime(c, j) = sngCompTime(c, j)
  Next j
Next c

'Find out the maximun completion time for each product
'and its value is stored in sngBuffTime(1, j)
For j = 1 To TJ
 For c = 2 To TC
  If sngBuffTime(c, j) > sngBuffTime(1, j) Then
   sngBuffTime(1, j) = sngBuffTime(c, j)
  End If
 Next c
Next j

'calculating the difference between the completion time of each job
'and that of the last job for each produbct variant
For j = 1 To TJ
 For c = 1 To TC
  sngDeltTime(c, j) = sngBuffTime(1, j) - sngCompTime(c, j)
  sngTotalDelt = sngTotalDelt + sngDeltTime(c, j)
```

```
  Next c
Next j
'calculating the WIP holding cost
sngWIP = sngUW * sngTotalDelt
'Objective value
sngObjective = sngMIC + sngMHC + sngCFC + sngWIP

End Function
```

'Progam Name: Solving the Combined Modular Product Scheduling and Production Cell
' Reconfiguration Problem: a GA Approach with Parallel Chromosome
' Coding
'Programmer: Hegui Ye
'Date: March 08,2005
'Decription: This form is mainly for inputting product and manufacturing systems
' data and input methods selection
'Folder: "C:\Documents and Settings\HEGUI YE\My Documents\GA\



Option Explicit

Dim TM As Integer, TV As Integer, MaxTI As Integer, TI() As Integer
Dim m As Integer, v As Integer, c As Integer, j As Integer, f As Integer
Dim strMsg As String, Demand() As Integer, varResponse As Variant

Private Sub Form_Load()
Load frmInput
frmInput.Show
End Sub

Private Sub txtTotalNumOfProducts_Change()
TV = CInt(txtTotalNumOfProducts.Text)
TJ = TV
End Sub

Private Sub txtTotalNumOfModules_Change()

```
TM = CInt(txtTotalNumOfModules.Text)
TC = TM
End Sub
```

```
Private Sub txtMaxNumOfInstances_Change()
MTI = CInt(txtMaxNumOfInstances.Text)
End Sub
```

```
Private Sub cmdContinue1_Click()
frmSelectionMethod.Enabled = True
chkInputData1.Enabled = True
chkInputData2.Enabled = True
End Sub
```

```
Private Sub chkInputData1_Click()
Call Manual_Selection
End Sub
```

```
Private Sub chkInputData2_Click()
Call GA_Selection
Load frmGAInformation
frmGAInformation.Show
Unload frmInput
frmInput.Hide
End Sub
```

```
Private Sub GA_Selection()

Open "C:\Documents and Settings\HEGUI YE\My Documents\GA\Data2-GA.txt" For
Output As #10

Dim MaxDemand As Integer, MinDemand As Integer, MaxFFC As Integer, MinFFC As
Integer
Dim MaxUMH As Integer, MinUMH As Integer, MaxUMI As Integer, MinUMI As
Integer
Dim MaxUPT As Integer, MinUPT As Integer
Dim F1 As Integer, F2 As Integer, j1 As Integer, j2 As Integer, i As Integer
Dim ModuleFEA() As Integer, InstSelection() As Integer, Demand() As Integer
Dim sngMFEA As Single

ReDim TI(TM), TF(TM), Q(TC, TJ), Demand(TV)
ReDim InstSelection(TM, TV), FFC(TC, MTI, MTI), ModuleFEA(TM, MTI, MTI)
```

```
ReDim sngUMH(TC, TJ, MTI), sngUMI(TC, TJ, MTI), sngUPT(TC, TJ, MTI),
FEA(TC, TJ, MTI)

MinDemand = 200
MaxDemand = 1000
MinFFC = 20
MaxFFC = 100
MaxUMH = 50
MinUMH = 2
MaxUMI = 20
MinUMI = 1
MaxUPT = 50
MinUPT = 5
'randomly generate an integer between 1 and MTL
'for the total instance number in each module
Write #10,
Write #10, "Number of instances in each module"
Write #10,
For m = 1 To TM
  TI(m) = Int(Rnd() * MTI) + 1
  TF(m) = TI(m)
  'MsgBox "TI(" & m & ") = " & TI(m)& "TF(" & m & ") = " & TF(m)
Next m
For m = 1 To TM
  Write #10, "TI( " & m & " ) = " & TI(m)
Next m
For m = 1 To TM
  Write #10, "TF( " & m & " ) = " & TF(m)
Next m

'randomly generate an integer between MinDemand and MaxDemand
'for each product variant
Write #10,
Write #10, "Product demand"
For v = 1 To TV
  Demand(v) = Int(Rnd() * (MaxDemand - MinDemand)) + MinDemand
  'MsgBox "Demand(" & v & ") = " & Demand(v)
  Write #10, "Demand( " & v & " ) = " & Demand(v)
Next v
'Calculating the production volume for each job in each cell
'Write #10,
'Write #10, "Job volume in each cell"
For c = 1 To TC
  For j = 1 To TJ
    Q(c, j) = Demand(j)
    'MsgBox "Job volume Q(" & c & "," & j & ") = " & Q(c, j)
```

```
    'Write #10, "Q( " & c & "," & j & " ) = " & Q(c, j)
  Next j
  'Write #10,
Next c

'randomly select an instance from each module for each product variant
For m = 1 To TM
  For v = 1 To TV
    InstSelection(m, v) = Int(Rnd() * TI(m)) + 1
  Next v
Next m

'Module-Configuration feasibility matrix
For m = 1 To TM
  For i = 1 To TI(m)
    For f = 1 To TF(m)
      If f = i Then
        ModuleFEA(m, i, f) = 1 'Guarantee one job has at least its own feasible conf
      Else
        sngMFEA = Round(Rnd(), 2)
        If sngMFEA >= 0.8 Or sngMFEA <= 0.2 Then
          ModuleFEA(m, i, f) = 1
        Else
          ModuleFEA(m, i, f) = 0
        End If
      End If
    Next f
  Next i
Next m

'Job-Configuration feasibility matrix is derived from
'the module-Configuration feasibility matrix
Write #10,
Write #10, "Job-Configuration feasibility matrix"
For c = 1 To TC
  For f = 1 To TF(c)
    For j = 1 To TJ
      For i = 1 To TI(c)
        If InstSelection(c, j) = i Then
          FEA(c, j, f) = ModuleFEA(c, i, f)
          Write #10, "FEA( " & c & "," & j & "," & f & " ) = " & FEA(c, j, f)
          Exit For
        End If
      Next i
    Next j
  Next f
```

```
  Write #10,
Next c

'Configuration-Configuration cost matrix
Write #10,
Write #10, "Configuration-Configuration cost matrix"
For c = 1 To TC
  For F1 = 1 To TF(c)
    For F2 = 1 To TF(c)
      If F2 = F1 Then
        FFC(c, F1, F2) = 0
      Else
        FFC(c, F1, F2) = Fix(Rnd() * (MaxFFC - MinFFC) + MinFFC)
      End If
      Write #10, "FFC( " & c & "," & F1 & "," & F2 & " ) = " & FFC(c, F1, F2)
    Next F2
  Next F1
  Write #10,
Next c

'Number of units of material handling cost matrix
Write #10,
Write #10, "Number of units of material handling cost matrix"
For c = 1 To TC
  For j = 1 To TJ
    For f = 1 To TF(c)
      If FEA(c, j, f) = 1 Then
        sngUMH(c, j, f) = Fix(Rnd() * (MaxUMH - MinUMH) + MinUMH)
      Else
        sngUMH(c, j, f) = BigM
      End If
      Write #10, "sngUMH( " & c & "," & j & "," & f & " ) = " & sngUMH(c, j, f)
    Next f
  Next j
  Write #10,
Next c

'Number of units of machine idle running cost matrix
Write #10,
Write #10, "Number of units of machine idle running cost matrix"
For c = 1 To TC
  For j = 1 To TJ
    For f = 1 To TF(c)
      If FEA(c, j, f) = 1 Then
        sngUMI(c, j, f) = Fix(Rnd() * (MaxUMI - MinUMI) + MinUMI)
      Else
```

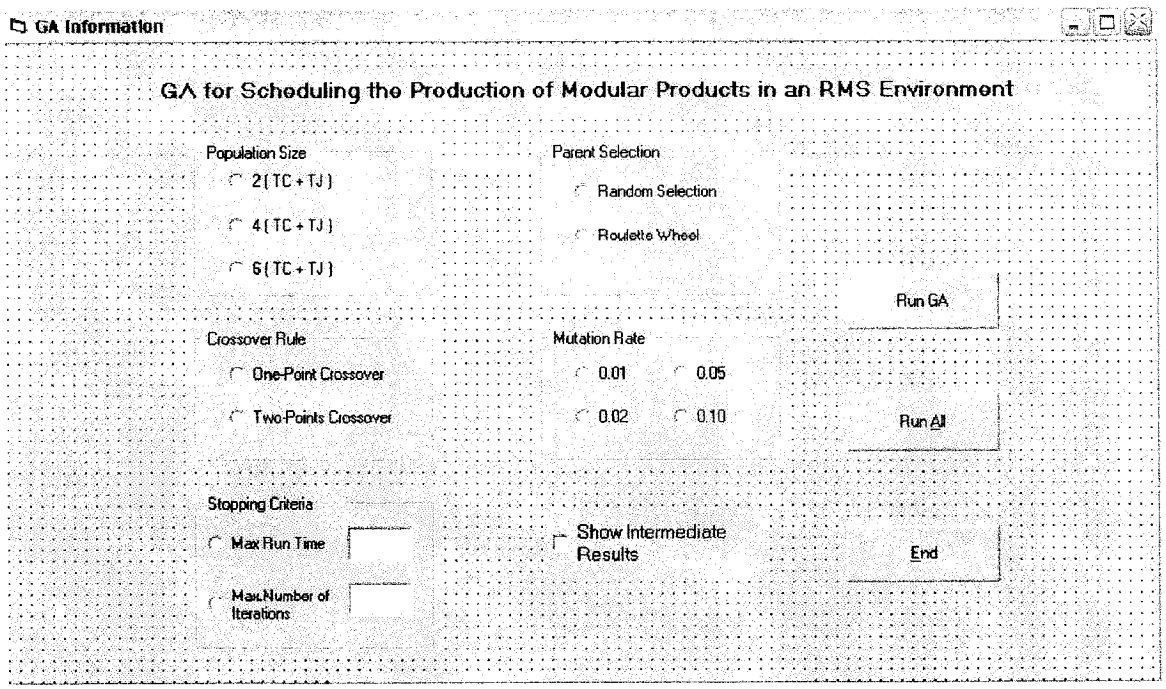
```
      sngUMI(c, j, f) = BigM
    End If
    Write #10, " sngUMI( " & c & "," & j & "," & f & " ) = " & sngUMI(c, j, f)
   Next f
  Next j
  Write #10,
Next c

'unit processing time matrix
Write #10,
Write #10, "unit processing time matrix"
For c = 1 To TC
  For j = 1 To TJ
   For f = 1 To TF(c)
    If FEA(c, j, f) = 1 Then
     sngUPT(c, j, f) = Fix(Rnd() * (MaxUPT - MinUPT) + MinUPT)
    Else
     sngUPT(c, j, f) = BigM
    End If
    Write #10, "sngUPT( " & c & "," & j & "," & f & " ) = " & sngUPT(c, j, f)
   Next f
  Next j
  Write #10,
Next c
End Sub
```

```
Private Sub Manual_Selection()

If TJ = 4 And TC = 5 Then
  Open "C:\Documents and Settings\HEGUI YE\My Documents\GA\data2-V4C5.txt" For
Input As #2
  MTI = TV
End If
If TJ = 4 And TC = 4 Then
  Open "C:\Documents and Settings\HEGUI YE\My Documents\GA\data2-V4C4.txt" For
Input As #2
  MTI = TV
End If
If TJ = 4 And TC = 3 Then
  Open "C:\Documents and Settings\HEGUI YE\My Documents\GA\data2-V4C3.txt" For
Input As #2
  MTI = TV
End If
If TJ = 3 And TC = 4 Then
```

```
  Open "C:\Documents and Settings\HEGUI YE\My Documents\GA\data2-V3C4.txt" For
Input As #2
  MTI = TV
End If
If TJ = 3 And TC = 5 Then
  Open "C:\Documents and Settings\HEGUI YE\My Documents\GA\data2-V3C5.txt" For
Input As #2
  'MTI = TV
End If

  'CC = CELLS; JJ = JOB; FF = CONFIGURATIONS
  Dim Demand() As Integer, CC As Integer, JJ As Integer, FF As Integer
  Dim strMsg As String, strTitle As String, F1 As Integer, F2 As Integer

  ReDim Demand(TJ), Q(TC, TJ)
  ReDim FEA(TC, TJ, MTI), FFC(TC, MTI, MTI)
  ReDim sngUMH(TC, TJ, MTI), sngUMI(TC, TJ, MTI)
  ReDim sngUPT(TC, TJ, MTI)
  ReDim TF(TC)

  'Input the number of configurations in each cell
  For CC = 1 To TC
    Input #2, TF(CC)
  Next CC
  'Product Demand
  For JJ = 1 To TJ
     Input #2, Demand(JJ)
  Next JJ
  'Calculatitng the job volume in each cell
  For CC = 1 To TC
    For JJ = 1 To TJ
      Q(CC, JJ) = Demand(JJ)
    Next JJ
  Next CC
  'Configuration-Module instance feasibility matrix in each cell
  For CC = 1 To TC
    For JJ = 1 To TJ
      For FF = 1 To TF(CC)
        Input #2, FEA(CC, JJ, FF)
      Next FF
    Next JJ
  Next CC
  'Configuration-configuration cost in each cell
  For CC = 1 To TC
    For F1 = 1 To TF(CC)
      For F2 = 1 To TF(CC)
```

```
      Input #2, FFC(CC, F1, F2)
    Next F2
  Next F1
Next CC
'Number of units of material handling if configuration
'F is used for job J in cell C
For CC = 1 To TC
  For JJ = 1 To TJ
    For FF = 1 To TF(CC)
      Input #2, sngUMH(CC, JJ, FF)
    Next FF
  Next JJ
Next CC
'Number of units of idl-running machine if configuration
'F is used for job J in cell C
For CC = 1 To TC
  For JJ = 1 To TJ
    For FF = 1 To TF(CC)
      Input #2, sngUMI(CC, JJ, FF)
    Next FF
  Next JJ
Next CC
'Number of units of WIP if configuration F is used for job J in cell C
For CC = 1 To TC
  For JJ = 1 To TJ
    For FF = 1 To TF(CC)
      Input #2, sngUPT(CC, JJ, FF)
    Next FF
  Next JJ
Next CC

Close #2

strMsg = "Data are all loaded. Choose <OK> to display GA information."
strTitle = "Data Loaded"
MsgBox strMsg, vbOKOnly & vbExclamation, strTitle
If vbOK Then
Load frmGAInformation
frmGAInformation.Show
End If

End Sub
```

'Progam Name: Solving the Combined Modular Product Scheduling and Production Cell
' Reconfiguration Problem: a GA Approach with Parallel Chromosome
' Coding
'Programmer: Hegui Ye
'Date: March 08,2005
'Decription: This form is used to select the GA parameter
'Folder: "C:\Documents and Settings\HEGUI YE\My Documents\GA\



Option Explicit

Private Sub chkCrossoverOption_Click(Index As Integer)
'one-point cutting
If chkCrossoverOption(0) = True Then
  CrossRule = 1
End If
'two-points cutting
If chkCrossoverOption(1) = True Then
  CrossRule = 2
End If
End Sub

Private Sub chkMutationOption_Click(Index As Integer)
If chkMutationOption(0) = True Then
  sngMutRate = 0.01

```
End If
If chkMutationOption(1) = True Then
  sngMutRate = 0.02
End If
If chkMutationOption(2) = True Then
  sngMutRate = 0.05
End If
If chkMutationOption(3) = True Then
  sngMutRate = 0.1
End If
End Sub
```

```
Private Sub chkParentSelection_Click(Index As Integer)
If chkParentSelection(0) = True Then
  ParentChoice = 1
End If
If chkParentSelection(1) = True Then
  ParentChoice = 2
End If
End Sub
```

```
Private Sub chkPopOption_Click(Index As Integer)
If chkPopOption(0) = True Then
  PopSize = 2 * (TC + TJ)
End If
If chkPopOption(1) = True Then
  PopSize = 4 * (TC + TJ)
End If
If chkPopOption(2) = True Then
  PopSize = 6 * (TC + TJ)
End If
End Sub
```

```
Private Sub chkShowResults_Click()
Load frmFinalResults
frmFinalResults.Show
frmFinalResults.Refresh
End Sub
```

```
Private Sub chkTerminalOption_Click(Index As Integer)
If chkTerminalOption(0) = True Then
   MaxRunTimeFlag = True
   txtMaxNumIteration.Text = ""
```

```
  txtMaxRunTime.Enabled = True
  txtMaxRunTime.Text = lngMaxRunTime
End If
If chkTerminalOption(1) = True Then
  MaxIterationFlag = True
  txtMaxRunTime.Text = ""
  txtMaxNumIteration.Enabled = True
  txtMaxNumIteration.Text = lngMaxIter
End If
End Sub
```

```
Private Sub Form_Load()
lngMaxIter = 20 * (TC + TJ + MTI)
lngMaxRunTime = 3600
End Sub
Private Sub cmdEnd_Click()
End
End Sub
```

```
Private Sub cmdRunAll_Click()
Dim Pa As Integer, Po As Integer, Mu As Integer, Cr As Integer, MaxCr As Integer
Dim sngTemp As Single, sngCurrentObject() As Single
Dim response As String
ReDim sngCurrentObject(2, 3, 2, 4)

Open "C:\Documents and Settings\HEGUI YE\My Documents\GA\Data1-Runall.txt" For
Output As #11

If chkTerminalOption(0) = False And chkTerminalOption(1) = False Then
  chkTerminalOption(1) = True
  MaxIterationFlag = True
  txtMaxRunTime.Text = ""
  txtMaxNumIteration.Enabled = True
  txtMaxNumIteration.Text = lngMaxIter
  response = MsgBox("GA stopping criterion is Maximum iteration number, " & _
       Chr(13) & "Do you want to change it? ", vbYesNo + vbQuestion + _
       vbDefaultButton2, "Information")
 If response = vbYes Then
   chkTerminalOption(0) = True
   MaxRunTimeFlag = True
   txtMaxNumIteration.Text = ""
   txtMaxRunTime.Enabled = True
   txtMaxRunTime.Text = lngMaxRunTime
  End If
```

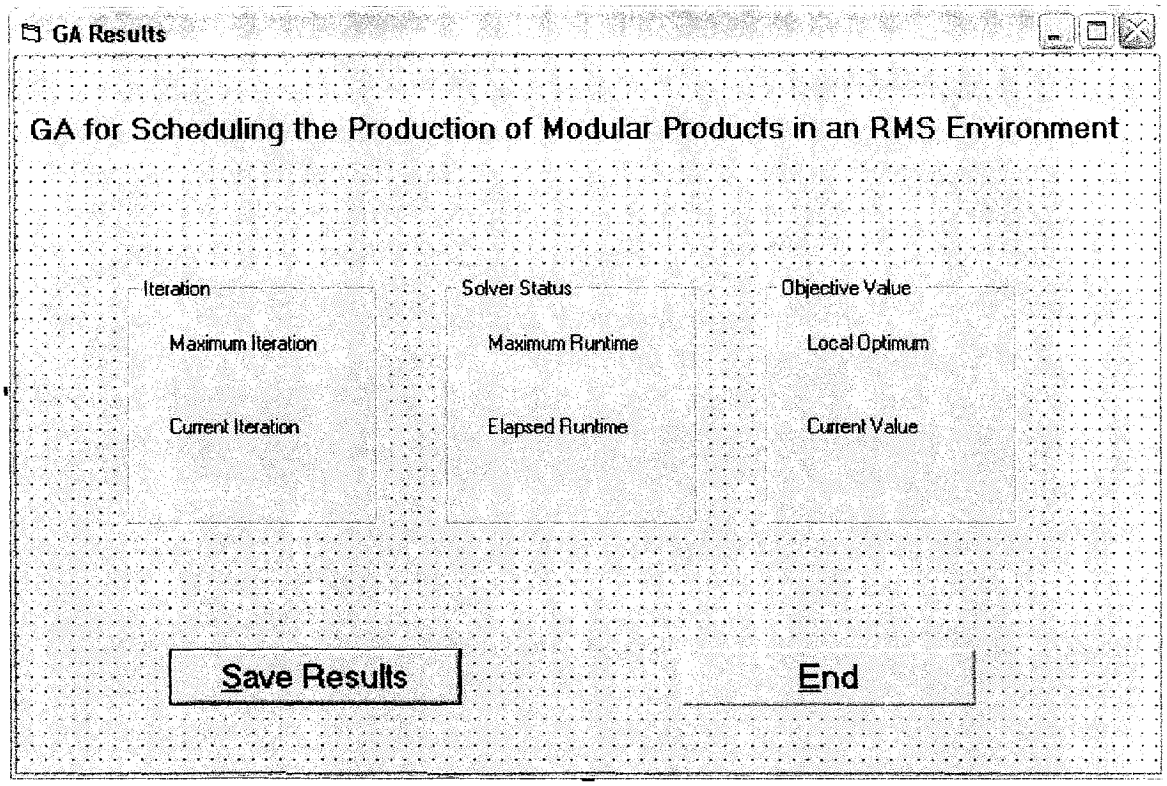
```
End If
lblStatus.Caption = "Calculating..."
lblStatus.Refresh
MaxCr = 2
lngStartTime = 0
lngElapseTime = 0
lngStartTime = Hour(Time) * 3600 + Minute(Time) * 60 + Second(Time)
For Pa = 1 To 2
 ParentChoice = Pa
 For Po = 1 To 3
  If Po = 1 Then
   PopSize = 2 * (TC + TJ)
  End If
  If Po = 2 Then
   PopSize = 4 * (TC + TJ)
  End If
  If Po = 3 Then
   PopSize = 6 * (TC + TJ)
  End If
   For Cr = 1 To MaxCr
    If Cr = 1 Then
     CrossRule = 1
    End If
    If Cr = 2 Then
     CrossRule = 2
    End If
     For Mu = 1 To 4
      If Mu = 1 Then
       sngMutRate = 0.01
      End If
      If Mu = 2 Then
       sngMutRate = 0.02
      End If
      If Mu = 3 Then
       sngMutRate = 0.05
      End If
      If Mu = 4 Then
       sngMutRate = 0.1
      End If

     Call GA
     Write #11, ParentChoice, PopSize, CrossRule, sngMutRate
     Write #11, sngFitness(1)
     Write #11,
     sngCurrentObject(Pa, Po, Cr, Mu) = sngFitness(1)
    Next Mu
```

```
    Next Cr
  Next Po
Next Pa
'calculate the minimum value among sngCurrentObject(Pa, Po, Cr, Mu)
For Pa = 1 To 2
  For Po = 1 To 3
    For Cr = 1 To MaxCr
      For Mu = 2 To 4
        If sngCurrentObject(Pa, Po, Cr, Mu) < sngCurrentObject(1, 1, 1, 1) Then
          sngTemp = sngCurrentObject(Pa, Po, Cr, Mu)
          sngCurrentObject(Pa, Po, Cr, Mu) = sngCurrentObject(1, 1, 1, 1)
          sngCurrentObject(1, 1, 1, 1) = sngTemp
        End If
      Next Mu
    Next Cr
  Next Po
Next Pa

'Calculate elapsed time GA has used
lngElapseTime = Hour(Time) * 3600 + Minute(Time) * 60 + Second(Time) _
- lngStartTime
'if the program runs overnight
If lngElapseTime < 0 Then
  lngElapseTime = (Hour(Time) + 24) * 3600 + Minute(Time) * 60 + _
  Second(Time) - lngStartTime
End If

Load frmFinalResults
frmFinalResults.Show
With frmFinalResults
    .lblValueOfMaxIter.Caption = lngMaxIter
    .lblValueOfIterate.Caption = lngNumIter
    .lblValueOfMaxTime.Caption = lngMaxRunTime
    .lblValueOfRunTime.Caption = lngElapseTime
    .lblValueOfOptimum.Caption = Round(sngFitness(1), 3)
    .lblValueOfCurrent.Caption = Round(sngCurrentObject(1, 1, 1, 1), 3)
End With
frmFinalResults.Refresh
Close #11
End Sub
```

```
Private Sub cmdRunGA_Click()

If chkTerminalOption(0) = False And chkTerminalOption(1) = False Then
  chkTerminalOption(1) = True
```

```
  MaxIterationFlag = True
  txtMaxRunTime.Text = ""
  txtMaxNumIteration.Enabled = True
  txtMaxNumIteration.Text = lngMaxIter
End If
lblStatus.Caption = "Calculating..."
lblStatus.Refresh
Call GA

Load frmFinalResults
frmFinalResults.Show
With frmFinalResults
    .lblValueOfMaxIter.Caption = lngMaxIter
    .lblValueOfIterate.Caption = lngNumIter
    .lblValueOfMaxTime.Caption = lngMaxRunTime
    .lblValueOfRunTime.Caption = lngElapseTime1
    .lblValueOfOptimum.Caption = Round(sngFitness(1), 3)
    .lblValueOfCurrent.Caption = Round(sngFitness(1), 3)
End With
frmFinalResults.Refresh
End Sub
```

'Progam Name: Solving the Combined Modular Product Scheduling and Production Cell
' Reconfiguration Problem: a GA Approach with Parallel Chromosome
' Coding
'Programmer: Hegui Ye
'Date: March 08,2005
'Decription: This form lists the computational results
'Folder: "C:\Documents and Settings\HEGUI YE\My Documents\GA\




```
Option explicit

Private Sub cmdEnd_Click()
   frmFinalResults.Hide
   Unload frmFinalResults
   End
End Sub
```

```
Private Sub cmdSave_Click()
  Open "C:\Documents and Settings\HEGUI YE\My Documents\GA\Data1.txt" For
Output As #1
```

```
Dim strMsg As String, strTitle As String
Dim blnResponse As Boolean

Write #1,
Write #1, "GA for Scheduling the Production of Modular Products in RMS
Environment"
Write #1,
Write #1,
Write #1, "Parent Selection = " & ParentChoice
Write #1, "Population Size = " & PopSize
Write #1, "Crossover Rule = " & CrossRule
Write #1, "Mutation Rate = " & sngMutRate
Write #1, "Crossover Rate = " & sngCrossRate
Write #1,
Write #1, "Stopping Critirion-Maximum Iteration = " & lngMaxIter
Write #1, "Total Iteration Number = " & lngNumIter
Write #1, "Total Elapsed Runtime = " & lngElapseTime
Write #1, "Last Iteration Elapsed Runtime = " & lngElapseTime1
Write #1,
Write #1, "Objective Value = " & sngFitness(1)
Write #1,
Write #1, "Job-Position assignments"

For c = 1 To TC
  For j = 1 To TJ
    For pp = 1 To TJ
     If X(c, j, pp) = 1 Then
       Write #1, "X(" & c & "," & j & "," & pp & ")= " & X(c, j, pp)
     Else
       X(c, j, pp) = 0
     End If
    Next pp
   Next j
   Write #1,
  Next c

Write #1,
Write #1, "Job-Configuration selection"
For c = 1 To TC
  For j = 1 To TJ
    For f = 1 To TF(c)
     If Z(c, j, f) = 1 Then
       Write #1, "Z(" & c & "," & j & "," & f & ")= " & Z(c, j, f)
     End If
    Next f
  Next j
```

```
  Write #1,
 Next c

 Close #1

 strMsg = "Data saved. Choose <End> botton to exit the program?"
 strTitle = "Save data"
 blnResponse = MsgBox(strMsg, vbOKOnly + vbQuestion + vbDefaultButton1, strTitle)

End Sub
```

```
Private Sub Form_Load()

 lblValueOfMaxIter.Caption = 0
 lblValueOfIterate.Caption = 0
 lblValueOfMaxTime.Caption = 0
 lblValueOfRunTime.Caption = 0
 lblValueOfOptimum.Caption = 0
 lblValueOfCurrent.Caption = 0
End Sub
```

Appendix C

Visual Basic 6.0 Code for GA with Serial Chromosome Coding

```
'Progam Name: Solving the Combined Modular Product Scheduling and Production Cell
'             Reconfiguration Problem: a GA Approach with Serial Chromosome
'             Coding
'Programmer:  Hegui Ye
'Date:        February 08,2005
'Decription:  This project aims at finding the minimum production cost of
'             producing a family of modular products,provided the structure of
'             modular products and the manufacturing systems are known.
'             This project incorpoates the following production cost elements:
'             unit processing time, material handling cost, machine relocation cost,
'             machine idle time costs, and work-in-process holding costs.
'Folder:      "C:\Documents and Settings\HEGUI YE\My Documents\GA2\

Option Explicit

Public Const sngUM = 0.02
Public Const sngUI = 0.003
Public Const sngUW = 0.05
Public Const BigM = 1000

'input variables
Public TC As Integer, TJ As Integer, MTI As Integer, TF() As Integer
Public Q() As Integer, FEA() As Integer, FFC() As Integer
Public sngUMH() As Integer, sngUMI() As Integer, sngUPT() As Single

'output variables
Public X() As Integer, Z() As Integer

'Results-form related variables
Public lngMaxRunTime As Long, lngElapseTime As Long, lngMaxIter As Long,
lngNumIter As Long
Public sngBestObject As Single, sngMutRate As Single, sngStartUp As Single,
lngStartTime As Long
Public ParentChoice As Integer, CrossRule As Integer, blnFlag As Boolean

'Chromosome related variables
Public ParentOne As Integer, ParentTwo As Integer, d As Integer, e As Integer
Public PopSize As Integer, Gene() As Integer, sngFitness() As Single

Sub GA()
```

```
'Temparary variables used for buffering data: BuffGen--for gene, sngBuffFit--for Fitness
Dim BuffGen As Integer, sngBuffFit As Single

Dim clnTempSet As New Collection, clnFinalSet As New Collection


Dim sngStartUp As Single

'Index variables: c--cell, j--job, p--population, pp--job position, f--configuration
'                i--collection item only, k--offspring index, a and b--temporary index
Dim a As Integer, b As Integer, c As Integer, f As Integer
Dim i As Integer, j As Integer, k As Integer, p As Integer, pp As Integer

ReDim sngFitness(3 * PopSize)
ReDim Gene(3 * PopSize, 2 * TC * TJ)
ReDim X(TC, TJ, TJ), Z(TC, TJ, MTI)

Open "C:\Documents and Settings\HEGUI YE\My Documents\GA2\Data3-Time.txt" For
Output As #9
Open "C:\Documents and Settings\HEGUI YE\My Documents\GA2\Data3-Fitness.txt"
For Output As #8
'Initialization
lngElapseTime = 0
lngNumIter = 0
lngNumIter = 0

sngStartUp = 999999999
'set timer
'lngStartTime = 0
'lngElapseTime = 0
'lngStartTime = Hour(Time) * 3600 + Minute(Time) * 60 + Second(Time)

'Generate initial population
'genes representing job-position assignments
Set clnTempSet = Nothing
Set clnFinalSet = Nothing
For p = 1 To PopSize
  For c = 1 To TC
    For j = 1 To TJ
      Do
        Gene(p, (2 * c - 2) * TJ + j) = Int(TJ * Rnd()) + 1
        If j = 1 Then
          clnFinalSet.Add Item:=Gene(p, (2 * c - 2) * TJ + j)
          Exit Do
        End If
        If j >= 2 Then
```

```
        For i = 1 To clnFinalSet.Count
          If Gene(p, (2 * c - 2) * TJ + j) <> clnFinalSet.Item(i) Then
            clnTempSet.Add Item:=Gene(p, (2 * c - 2) * TJ + j)
          End If
        Next i
        If clnTempSet.Count = clnFinalSet.Count Then
          clnFinalSet.Add Item:=Gene(p, (2 * c - 2) * TJ + j)
        End If
        Set clnTempSet = Nothing
       End If
     Loop Until clnFinalSet.Count = j
   Next j
   For j = 1 To TJ
     Gene(p, (2 * c - 2) * TJ + j) = clnFinalSet.Item(j)
   Next j
   Set clnFinalSet = Nothing
  Next c
Next p
'genes representing job-configuration selection
For p = 1 To PopSize
  For c = 1 To TC
    For j = 1 To TJ
      Do
        Gene(p, (2 * c - 1) * TJ + j) = Int(TF(c) * Rnd()) + 1
      Loop Until FEA(c, j, Gene(p, (2 * c - 1) * TJ + j)) = 1
    Next j
  Next c
Next p

'calculating objective function value and assign it as fitness value
For p = 1 To PopSize
sngFitness(p) = sngObjective(p)
Next p

'iterate until meet the stopping criterion
'i.e. lngNumIter = lngMaxIter(maximum iteration number)
'or Elapsed run time = Maximum run time
Write #9, "Elapsed Run Time = "
Write #8, "Objective = "
Do
  d = 0
  e = 0
  'iterate to generate offsprings until the number of offsprings = Popsize
  Do
    Select_Parents
    CrossOver
```

```
Loop While d + e < PopSize

'call sub mutation
Mutation

'compare the first batch of offsprings with their parent chromosomes,
'if their fitness value is smaller than that of their parent chromosomes,
'they will replace parent chromosomes.
For p = 1 To PopSize
  For k = 1 To d
    'compare the fitness values of the first batch of offsprings with that of
    'each of their parents,if smaller, the offspring chromosomes will survive
    If sngFitness(PopSize + k) < sngFitness(p) Then
      sngBuffFit = sngFitness(PopSize + k)
      sngFitness(PopSize + k) = sngFitness(p)
      sngFitness(p) = sngBuffFit
      'if offspring chromosome survives,the following code segments
      'exchange all genes between offsprings and their parents
      For c = 0 To 2 * TC - 1
        For j = 1 To TJ
          BuffGen = Gene(PopSize + k, c * TJ + j)
          Gene(PopSize + k, c * TJ + j) = Gene(p, c * TJ + j)
          Gene(p, c * TJ + j) = BuffGen
        Next j
      Next c
    End If
  Next k
Next p

'compare the second batch of offsprings with parent chromosomes,
'if their fitness value is smaller than that of their parent chromosomes,
'they will replace parent chromosomes.
For p = 1 To PopSize
  For k = 1 To e
    'compare the fitness values of the second batch of offsprings with that of
    'each of their parents,if smaller, the offspring chromosomes will survive
    If sngFitness(PopSize * 2 + k) < sngFitness(p) Then
      sngBuffFit = sngFitness(PopSize * 2 + k)
      sngFitness(PopSize * 2 + k) = sngFitness(p)
      sngFitness(p) = sngBuffFit
      'if offspring chromosome survives,the following code segments
      'exchange all genes between offsprings and their parents
      For c = 0 To 2 * TC - 1
        For j = 1 To TJ
          BuffGen = Gene(PopSize * 2 + k, c * TJ + j)
          Gene(PopSize * 2 + k, c * TJ + j) = Gene(p, c * TJ + j)
```

```
          Gene(p, c * TJ + j) = BuffGen
        Next j
      Next c
    End If
  Next k
Next p

'compare the fitness values of current populations,
'the smallest one is retained in chromosome nnumber one
For p = 2 To PopSize
  If sngFitness(p) < sngFitness(1) Then
    sngBuffFit = sngFitness(p)
    sngFitness(p) = sngFitness(1)
    sngFitness(1) = sngBuffFit
    For c = 0 To 2 * TC - 1
      For j = 1 To TJ
        BuffGen = Gene(p, c * TJ + j)
        Gene(p, c * TJ + j) = Gene(1, c * TJ + j)
        Gene(1, c * TJ + j) = BuffGen
      Next j
    Next c
  End If
Next p

'Display elapsed time GA has used
'lngElapseTime = Hour(Time) * 3600 + Minute(Time) * 60 + Second(Time) -
lngStartTime
'if the program runs overnight
'If lngElapseTime < 0 Then
  'lngElapseTime = (Hour(Time) + 24) * 3600 + Minute(Time) * 60 + Second(Time) -
lngStartTime
'End If

'check the smallest fitness value with startup value,if satisfied,
'print out the job-position assignments and job configuration selections
If sngFitness(1) < sngStartUp Then
  'job-position assignments
  For p = 1 To PopSize
    For c = 1 To TC
      For j = 1 To TJ
        For pp = 1 To TJ
          If Gene(p, (2 * c - 2) * TJ + j) = pp Then
            X(c, j, pp) = 1
          Else
            X(c, j, pp) = 0
          End If
```

```
        Next pp
      Next j
    Next c
  Next p

  'job-configuration selection
  For p = 1 To PopSize
    For c = 1 To TC
      For j = 1 To TJ
        For f = 1 To TF(c)
          If Gene(p, (2 * c - 1) * TJ + j) = f Then
            Z(c, j, f) = 1
          Else
            Z(c, j, f) = 0
          End If
        Next f
      Next j
    Next c
  Next p
  'replace the startup value with the smallest fitness value obtained so far
  'this will force the GA to find a even smaller fitness value--better solution
  sngStartUp = sngFitness(1)
 End If
  lngNumIter = lngNumIter + 1
  sngBestObject = sngFitness(1)
  Write #9, lngElapseTime
  Write #8, sngBestObject
 Loop While lngNumIter < lngMaxIter
 Close #9
 Close #8
End Sub

Sub Select_Parents()

Dim p As Integer, sngRouWheel() As Single
Dim sngSumFit As Single, sngPointer1 As Single, sngPointer2 As Single
ReDim sngRouWheel(PopSize)

'Random parent selection
If ParentChoice = 1 Then
 Do
  ParentOne = Fix(PopSize * Rnd())
 Loop Until ParentOne <> 0
 Do
  ParentTwo = Fix(PopSize * Rnd())
 Loop While (ParentTwo = ParentOne Or ParentTwo = 0 Or sngFitness(ParentTwo) = 0)
```

```
End If

'Roullete wheel parent selection
If ParentChoice = 2 Then
  sngSumFit = 0
  sngRouWheel(0) = 0
  For p = 1 To PopSize
    sngSumFit = sngSumFit + 1 / sngFitness(p)
  Next p
  For p = 1 To PopSize
    sngRouWheel(p) = sngRouWheel(p - 1) + 1 / (sngFitness(p) * sngSumFit)
  Next p
  sngPointer1 = Rnd()
  For p = 1 To PopSize
    If sngRouWheel(p) > sngPointer1 Then Exit For
  Next p
  ParentOne = p
  Do
    sngPointer2 = Rnd()
    For p = 1 To PopSize
      If sngRouWheel(p) > sngPointer2 Then Exit For
    Next p
    ParentTwo = p
  Loop While ParentTwo = ParentOne
End If
End Sub

Sub CrossOver()

Dim c As Integer, j As Integer, i As Integer, BuffCut As Integer
Dim Cut1_Pos As Integer, Cut2_Pos As Integer, Cut1_Conf As Integer, Cut2_Conf As
Integer
Dim clnParOneSet As New Collection, clnParTwoSet As New Collection

Select Case CrossRule
Case 1  'one-one cutting
  'for small-size problem, one-point cut crossover is adopted.
  'generate cut point for genes representing job-position assignments
  For c = 1 To TC
    Cut1_Pos = Fix(TJ * Rnd())
    If Cut1_Pos = 0 Then
      Cut1_Pos = 1
    End If
    'retain genes before the one-point cut
    For j = 1 To Cut1_Pos
      'the first offspring retains genes from parent1 before the cut-point
```

```
    Gene(PopSize + d + 1, (2 * c - 2) * TJ + j) = Gene(ParentOne, (2 * c - 2) * TJ + j)
    'the second offspring retains genes from parent2 before the cut-point
    Gene(PopSize * 2 + e + 1, (2 * c - 2) * TJ + j) = Gene(ParentTwo, (2 * c - 2) * TJ + j)
  Next j

  'Offsprings' genes after the cut-point will follow the sequence of their counterpart
parents
  'i.e. genes after the cut-point in offspring1 follow the sequence of genes in parent2
  'and genes after the cut-point in offspring2 follow the sequence of genes in parent1

  'Compare each gene in parent2 with genes after cut point in parent1,if same,
  'add those genes in parent2 that match genes after cut point in parent1 to clnParOne.
  'so collection "clnParOne" includes all the genes after cut point in parent1,
  'but these genes have been reordered according to the gene sequence in parent2
  'If Cut1_Pos + 1 <= TJ - 1 Then
  Set clnParOneSet = Nothing
  For i = 1 To TJ
    For j = Cut1_Pos + 1 To TJ
      If Gene(ParentTwo, (2 * c - 2) * TJ + i) = Gene(ParentOne, (2 * c - 2) * TJ + j)
Then
        clnParOneSet.Add Item:=Gene(ParentTwo, (2 * c - 2) * TJ + i)
        Exit For
      End If
    Next j
  Next i

  For j = Cut1_Pos + 1 To TJ
    Gene(PopSize + d + 1, (2 * c - 2) * TJ + j) = clnParOneSet.Item(j - Cut1_Pos)
  Next j
  Set clnParOneSet = Nothing

  'Compare each gene in parent1 with genes after cut point in parent2,if same,
  'add those genes in parent1 that match the genes after cut point in parent1 to
clnParTwo.
  'so collection "clnParTwo" includes all the genes after cut point in parent2,
  'but these genes have been reordered according to the gene sequence in parent1
  Set clnParTwoSet = Nothing
  For i = 1 To TJ
    For j = Cut1_Pos + 1 To TJ
      If Gene(ParentOne, (2 * c - 2) * TJ + i) = Gene(ParentTwo, (2 * c - 2) * TJ + j)
Then
        clnParTwoSet.Add Item:=Gene(ParentOne, (2 * c - 2) * TJ + i)
        Exit For
      End If
    Next j
  Next i
```

```
  For j = Cut1_Pos + 1 To TJ
    Gene(PopSize * 2 + e + 1, (2 * c - 2) * TJ + j) = clnParTwoSet.Item(j - Cut1_Pos)
  Next j
  'End If
  Set clnParTwoSet = Nothing

  'generate cut point for genes representing job-configuration selections
  Cut1_Conf = Fix(TJ * Rnd())
  If Cut1_Conf = 0 Then
    Cut1_Conf = 1
  End If
  'retain genes before the one-point cut
  For j = 1 To Cut1_Conf
    'the first offspring retains genes from parent1 before the cut-point
    Gene(PopSize + d + 1, (2 * c - 1) * TJ + j) = Gene(ParentOne, (2 * c - 1) * TJ + j)
    'the second offspring retains genes from parent2 before the cut-point
    Gene(PopSize * 2 + e + 1, (2 * c - 1) * TJ + j) = Gene(ParentTwo, (2 * c - 1) * TJ + j)
  Next j

  'exchange genes after the one-point cut
  For j = Cut1_Conf + 1 To TJ
    'the first offspring gains genes from parent2 after the cut-point
    Gene(PopSize + d + 1, (2 * c - 1) * TJ + j) = Gene(ParentTwo, (2 * c - 1) * TJ + j)
    'the second offspring gains genes from parent1 after the cut-point
    Gene(PopSize * 2 + e + 1, (2 * c - 1) * TJ + j) = Gene(ParentOne, (2 * c - 1) * TJ + j)
  Next j
 Next c

Case 2  'one-two cutting
 'generate cut point for genes representing job-position assignments
 For c = 1 To TC
  Cut1_Pos = Fix(TJ * Rnd())
  If Cut1_Pos = 0 Then
    Cut1_Pos = 1
  End If
  'retain genes before the one-point cut
  For j = 1 To Cut1_Pos
    'the first offspring retains genes from parent1 before the cut-point
    Gene(PopSize + d + 1, (2 * c - 2) * TJ + j) = Gene(ParentOne, (2 * c - 2) * TJ + j)
    'the second offspring retains genes from parent2 before the cut-point
    Gene(PopSize * 2 + e + 1, (2 * c - 2) * TJ + j) = Gene(ParentTwo, (2 * c - 2) * TJ + j)
  Next j

  'Offsprings' genes after the cut-point will follow the sequence of their counterpart
parents
```

```
'i.e. genes after the cut-point in offspring1 follow the sequence of genes in parent2
'and genes after the cut-point in offspring2 follow the sequence of genes in parent1

'Compare each gene in parent2 with genes after cut point in parent1,if same,
'add those genes in parent2 that match genes after cut point in parent1 to clnParOne.
'so collection "clnParOne" includes all the genes after cut point in parent1,
'but these genes have been reordered according to the
'gene sequence in parent2
Set clnParOneSet = Nothing
For i = 1 To TJ
 For j = Cut1_Pos + 1 To TJ
  If Gene(ParentTwo, (2 * c - 2) * TJ + i) = Gene(ParentOne, (2 * c - 2) * TJ + j)
Then
    clnParOneSet.Add Item:=Gene(ParentTwo, (2 * c - 2) * TJ + i)
    Exit For
  End If
 Next j
Next i

For j = Cut1_Pos + 1 To TJ
 Gene(PopSize + d + 1, (2 * c - 2) * TJ + j) = clnParOneSet.Item(j - Cut1_Pos)
Next j

'Compare each gene in parent1 with genes after cut point in parent2,if same,
'add those genes in parent1 that match the genes after cut point in parent1 to
clnParTwo.
'so collection "clnParTwo" includes all the genes after cut point in parent2,
'but these genes have been reordered according to the gene sequence in parent1
Set clnParTwoSet = Nothing
For i = 1 To TJ
 For j = Cut1_Pos + 1 To TJ
  If Gene(ParentOne, (2 * c - 2) * TJ + i) = Gene(ParentTwo, (2 * c - 2) * TJ + j)
Then
    clnParTwoSet.Add Item:=Gene(ParentOne, (2 * c - 2) * TJ + i)
    Exit For
  End If
 Next j
Next i

For j = Cut1_Pos + 1 To TJ
 Gene(PopSize * 2 + e + 1, (2 * c - 2) * TJ + j) = clnParTwoSet.Item(j - Cut1_Pos)
Next j

'generate cut point for genes representing job-configuration selections
Cut1_Conf = Fix(TJ * Rnd())
If Cut1_Conf = 0 Then
```

```
    Cut1_Conf = 1
   End If
   Do
    Cut2_Conf = Fix(TJ * Rnd())
    If Cut2_Conf = 0 Then
      Cut2_Conf = TJ - 1
    End If
   Loop While Cut2_Conf = Cut1_Conf
   'exchange the value of Cut2_Conf and Cut1_Conf if Cut2_Conf < Cut1_Conf
   If Cut2_Conf < Cut1_Conf Then
    BuffCut = Cut1_Conf
    Cut1_Conf = Cut2_Conf
    Cut2_Conf = BuffCut
   End If
   'retain genes before the first cut point
   For j = 1 To Cut1_Conf
    'the first offspring retains genes from parent1 before the cut-point
    Gene(PopSize + d + 1, (2 * c - 1) * TJ + j) = Gene(ParentOne, (2 * c - 1) * TJ + j)
    'the second offspring retains genes from parent2 before the cut-point
    Gene(PopSize * 2 + e + 1, (2 * c - 1) * TJ + j) = Gene(ParentTwo, (2 * c - 1) * TJ + j)
   Next j
   'retain genes  after second cut point
   For j = Cut2_Conf To TJ
    'the first offspring retains genes from parent1 after the cut-point
    Gene(PopSize + d + 1, (2 * c - 1) * TJ + j) = Gene(ParentOne, (2 * c - 1) * TJ + j)
    'the second offspring retains genes from parent2 after the cut-point
    Gene(PopSize * 2 + e + 1, (2 * c - 1) * TJ + j) = Gene(ParentTwo, (2 * c - 1) * TJ + j)
   Next j

   'exchange genes between the first and second cut points
   If Cut1_Conf + 1 <= Cut2_Conf - 1 Then
    For j = Cut1_Conf + 1 To Cut2_Conf - 1
     'the first offspring gains genes from parent2 between the first and second cut points
     Gene(PopSize + d + 1, (2 * c - 1) * TJ + j) = Gene(ParentTwo, (2 * c - 1) * TJ + j)
     'the second offspring gains genes from parent1 between the first and second cut
points
     Gene(PopSize * 2 + e + 1, (2 * c - 1) * TJ + j) = Gene(ParentOne, (2 * c - 1) * TJ +
j)
    Next j
   End If
  Next c

Case 3  'two-one cutting
  'generate cut point for genes representing job-position assignments
  For c = 1 To TC
   Cut1_Pos = Fix(TJ * Rnd())
```

```
  If Cut1_Pos = 0 Then
    Cut1_Pos = 1
  End If
  Do
   Cut2_Pos = Fix(TJ * Rnd())
   If Cut2_Pos = 0 Then
     Cut2_Pos = TJ - 1
   End If
  Loop While Cut2_Pos = Cut1_Pos
  'exchange the value of Cut2_Pos and Cut1_Pos if Cut2_Pos < Cut1_Pos
  If Cut2_Pos < Cut1_Pos Then
   BuffCut = Cut1_Pos
   Cut1_Pos = Cut2_Pos
   Cut2_Pos = BuffCut
  End If
  'retain genes before the first cut point
  For j = 1 To Cut1_Pos
   'the first offspring retains genes from parent1 before the first cut-point
   Gene(PopSize + d + 1, (2 * c - 2) * TJ + j) = Gene(ParentOne, (2 * c - 2) * TJ + j)
   'the second offspring retains genes from parent2 before the first cut-point
   Gene(PopSize * 2 + e + 1, (2 * c - 2) * TJ + j) = Gene(ParentTwo, (2 * c - 2) * TJ + j)
  Next j
  'retain genes after the second cut point
  For j = Cut2_Pos To TJ
   'the first offspring retains genes from parent1 after the second cut-point
   Gene(PopSize + d + 1, (2 * c - 2) * TJ + j) = Gene(ParentOne, (2 * c - 2) * TJ + j)
   'the second offspring retains genes from parent2 after the second cut-point
   Gene(PopSize * 2 + e + 1, (2 * c - 2) * TJ + j) = Gene(ParentTwo, (2 * c - 2) * TJ + j)
  Next j

  'Offsprings' genes between the two cut-points will follow the sequence of their
counterpart parents
  'i.e. genes between the two cut-points in offspring1 follow the sequence of genes in
parent2
  'and genes between the two cut-points in offspring2 follow the sequence of genes in
parent1

  'Compare each gene in parent2 with genes between the two cut-points in parent1,if
same,
  'add those genes in parent2 that match genes between the two cut-points in parent1 to
clnParOne.
  'so collection "clnParOne" includes all the genes between the two cut-points in parent1,
  'but these genes have been reordered according to the
  'gene sequence in parent2
  If Cut1_Pos + 1 <= Cut2_Pos - 1 Then
   Set clnParOneSet = Nothing
```

```
    For i = 1 To TJ
      For j = Cut1_Pos + 1 To Cut2_Pos - 1
        If Gene(ParentTwo, (2 * c - 2) * TJ + i) = Gene(ParentOne, (2 * c - 2) * TJ + j)
Then
          clnParOneSet.Add Item:=Gene(ParentTwo, (2 * c - 2) * TJ + i)
          Exit For
        End If
      Next j
    Next i
    'restore genes value
    For j = Cut1_Pos + 1 To Cut2_Pos - 1
      Gene(PopSize + d + 1, (2 * c - 2) * TJ + j) = clnParOneSet.Item(j - Cut1_Pos)
    Next j

    'Compare each gene in parent1 with genes between the two cut-points in parent2,if
same,
    'add those genes in parent1 that match the genes between the two cut-points in
parent1 to clnParTwo.
    'so collection "clnParTwo" includes all the genes between the two cut-points in
parent2,
    'but these genes have been reordered according to the gene sequence in parent1
    Set clnParTwoSet = Nothing
    For i = 1 To TJ
      For j = Cut1_Pos + 1 To Cut2_Pos - 1
        If Gene(ParentOne, (2 * c - 2) * TJ + i) = Gene(ParentTwo, (2 * c - 2) * TJ + j)
Then
          clnParTwoSet.Add Item:=Gene(ParentOne, (2 * c - 2) * TJ + i)
          Exit For
        End If
      Next j
    Next i

    For j = Cut1_Pos + 1 To Cut2_Pos - 1
      Gene(PopSize * 2 + e + 1, (2 * c - 2) * TJ + j) = clnParTwoSet.Item(j - Cut1_Pos)
    Next j
  End If

  'generate cut point for genes representing job-configuration selections
  Cut1_Conf = Fix(TJ * Rnd())
  If Cut1_Conf = 0 Then
    Cut1_Conf = 1
  End If
  'retain genes before the one-point cut
  For j = 1 To Cut1_Conf
    'the first offspring retains genes from parent1 before the cut-point
    Gene(PopSize + d + 1, (2 * c - 1) * TJ + j) = Gene(ParentOne, (2 * c - 1) * TJ + j)
```

```
    'the second offspring retains genes from parent2 before the cut-point
    Gene(PopSize * 2 + e + 1, (2 * c - 1) * TJ + j) = Gene(ParentTwo, (2 * c - 1) * TJ + j)
   Next j

   'exchange genes after the one-point cut
   For j = Cut1_Conf + 1 To TJ
    'the first offspring gains genes from parent2 after the cut-point
    Gene(PopSize + d + 1, (2 * c - 1) * TJ + j) = Gene(ParentTwo, (2 * c - 1) * TJ + j)
    'the second offspring gains genes from parent1 after the cut-point
    Gene(PopSize * 2 + e + 1, (2 * c - 1) * TJ + j) = Gene(ParentOne, (2 * c - 1) * TJ + j)
   Next j
  Next c
Case 4  'two-two cutting
  'generate cut point for genes representing job-position assignments
  For c = 1 To TC
   Cut1_Pos = Fix(TJ * Rnd())
   If Cut1_Pos = 0 Then
     Cut1_Pos = 1
   End If
   Do
    Cut2_Pos = Fix(TJ * Rnd())
    If Cut2_Pos = 0 Then
      Cut2_Pos = TJ - 1
    End If
   Loop While Cut2_Pos = Cut1_Pos
   'exchange the value of Cut2_Pos and Cut1_Pos if Cut2_Pos < Cut1_Pos
   If Cut2_Pos < Cut1_Pos Then
    BuffCut = Cut1_Pos
    Cut1_Pos = Cut2_Pos
    Cut2_Pos = BuffCut
   End If
   'retain genes before the first cut point
   For j = 1 To Cut1_Pos
    'the first offspring retains genes from parent1 before the first cut-point
    Gene(PopSize + d + 1, (2 * c - 2) * TJ + j) = Gene(ParentOne, (2 * c - 2) * TJ + j)
    'the second offspring retains genes from parent2 before the first cut-point
    Gene(PopSize * 2 + e + 1, (2 * c - 2) * TJ + j) = Gene(ParentTwo, (2 * c - 2) * TJ + j)
   Next j
   'retain genes after the second cut point
   For j = Cut2_Pos To TJ
    'the first offspring retains genes from parent1 after the second cut-point
    Gene(PopSize + d + 1, (2 * c - 2) * TJ + j) = Gene(ParentOne, (2 * c - 2) * TJ + j)
    'the second offspring retains genes from parent2 after the second cut-point
    Gene(PopSize * 2 + e + 1, (2 * c - 2) * TJ + j) = Gene(ParentTwo, (2 * c - 2) * TJ + j)
   Next j
```

```
'Offsprings' genes between the two cut-points will follow the sequence of their
counterpart parents
'i.e. genes between the two cut-points in offspring1 follow the sequence of genes in
parent2
'and genes between the two cut-points in offspring2 follow the sequence of genes in
parent1

'Compare each gene in parent2 with genes between the two cut-points in parent1,if
same,
'add those genes in parent2 that match genes between the two cut-points in parent1 to
clnParOne.
'so collection "clnParOne" includes all the genes between the two cut-points in parent1,
'but these genes have been reordered according to the
'gene sequence in parent2
If Cut1_Pos + 1 <= Cut2_Pos - 1 Then
  Set clnParOneSet = Nothing
  For i = 1 To TJ
    For j = Cut1_Pos + 1 To Cut2_Pos - 1
      If Gene(ParentTwo, (2 * c - 2) * TJ + i) = Gene(ParentOne, (2 * c - 2) * TJ + j)
Then
        clnParOneSet.Add Item:=Gene(ParentTwo, (2 * c - 2) * TJ + i)
        Exit For
      End If
    Next j
  Next i
  'restore genes value
  For j = Cut1_Pos + 1 To Cut2_Pos - 1
    Gene(PopSize + d + 1, (2 * c - 2) * TJ + j) = clnParOneSet.Item(j - Cut1_Pos)
  Next j

  'Compare each gene in parent1 with genes between the two cut-points in parent2,if
same,
  'add those genes in parent1 that match the genes between the two cut-points in
parent1 to clnParTwo.
  'so collection "clnParTwo" includes all the genes between the two cut-points in
parent2,
  'but these genes have been reordered according to the gene sequence in parent1
  Set clnParTwoSet = Nothing
  For i = 1 To TJ
    For j = Cut1_Pos + 1 To Cut2_Pos - 1
      If Gene(ParentOne, (2 * c - 2) * TJ + i) = Gene(ParentTwo, (2 * c - 2) * TJ + j)
Then
        clnParTwoSet.Add Item:=Gene(ParentOne, (2 * c - 2) * TJ + i)
        Exit For
      End If
    Next j
```

```
    Next i
    For j = Cut1_Pos + 1 To Cut2_Pos - 1
      Gene(PopSize * 2 + e + 1, (2 * c - 2) * TJ + j) = clnParTwoSet.Item(j - Cut1_Pos)
    Next j
  End If

  'generate cut point for genes representing job-configuration selections
  Cut1_Conf = Fix(TJ * Rnd())
  If Cut1_Conf = 0 Then
    Cut1_Conf = 1
  End If
  Do
    Cut2_Conf = Fix(TJ * Rnd())
    If Cut2_Conf = 0 Then
      Cut2_Conf = TJ - 1
    End If
  Loop While Cut2_Conf = Cut1_Conf
  'exchange the value of Cut2_Conf and Cut1_Conf if Cut2_Conf < Cut1_Conf
  If Cut2_Conf < Cut1_Conf Then
    BuffCut = Cut1_Conf
    Cut1_Conf = Cut2_Conf
    Cut2_Conf = BuffCut
  End If
  'retain genes before the first cut point
  For j = 1 To Cut1_Conf
    'the first offspring retains genes from parent1 before the first cut-point
    Gene(PopSize + d + 1, (2 * c - 1) * TJ + j) = Gene(ParentOne, (2 * c - 1) * TJ + j)
    'the second offspring retains genes from parent2 before the first cut-point
    Gene(PopSize * 2 + e + 1, (2 * c - 1) * TJ + j) = Gene(ParentTwo, (2 * c - 1) * TJ + j)
  Next j
  'retain genes  after second cut point
  For j = Cut2_Conf To TJ
    'the first offspring retains genes from parent1 after the second cut-point
    Gene(PopSize + d + 1, (2 * c - 1) * TJ + j) = Gene(ParentOne, (2 * c - 1) * TJ + j)
    'the second offspring retains genes from parent2 after the second cut-point
    Gene(PopSize * 2 + e + 1, (2 * c - 1) * TJ + j) = Gene(ParentTwo, (2 * c - 1) * TJ + j)
  Next j

  'exchange genes between the first and second cut points
  If Cut1_Conf + 1 <= Cut2_Conf - 1 Then
    For j = Cut1_Conf + 1 To Cut2_Conf - 1
      'the first offspring gains genes from parent2 between the first and second cut points
      Gene(PopSize + d + 1, (2 * c - 1) * TJ + j) = Gene(ParentTwo, (2 * c - 1) * TJ + j)
      'the second offspring gains genes from parent1 between the first and second cut
points
```

```
      Gene(PopSize * 2 + e + 1, (2 * c - 1) * TJ + j) = Gene(ParentOne, (2 * c - 1) * TJ +
j)
    Next j
   End If
  Next c
End Select

d = d + 1
e = e + 1
sngFitness(PopSize + d) = sngObjective(PopSize + d)
sngFitness(PopSize * 2 + e) = sngObjective(PopSize * 2 + e)
End Sub
Sub Mutation()

Dim MutNum As Integer, NewP As Integer, k As Integer, c As Integer, j As Integer, b As
Integer
Dim MutPos1 As Integer, MutPos2 As Integer, OldGenVal() As Integer
Dim sngFitNew As Single, blnSearchFlag As Boolean

ReDim OldGenVal(2 * TC * TJ)

'randomly select MutNuM Chromosomes for mutation
MutNum = Int(sngMutRate * (PopSize * (TC + TJ) * Rnd())) + 1
For k = 1 To MutNum
 'randomly selecting one chromosome
 NewP = Int(PopSize * Rnd()) + 1
 'keep the fitness value before mutation
 sngFitNew = sngFitness(NewP)
 'keep the genes's values before mutation
 For c = 0 To 2 * TC - 1
  For j = 1 To TJ
   OldGenVal(c * TJ + j) = Gene(NewP, c * TJ + j)
  Next j
 Next c

 For c = 1 To TC
  'randomly select one mutation position for genes representing job-position assignment
  MutPos1 = Int(TJ * Rnd()) + 1
  Do
   'radomly generate a value for the newly selected gene,
   'it should not be the same as MutGenVal((2 * c - 2) * TJ + MutPos1)
   Gene(NewP, (2 * c - 2) * TJ + MutPos1) = Int(TJ * Rnd()) + 1
  Loop Until Gene(NewP, (2 * c - 2) * TJ + MutPos1) <> OldGenVal((2 * c - 2) * TJ +
MutPos1)

  If MutPos1 = 1 Then
```

```
    For b = 2 To TJ
     If Gene(NewP, (2 * c - 2) * TJ + MutPos1) = OldGenVal((2 * c - 2) * TJ + b) Then
      Gene(NewP, (2 * c - 2) * TJ + b) = OldGenVal((2 * c - 2) * TJ + MutPos1)
      Gene(NewP, (2 * c - 2) * TJ + MutPos1) = OldGenVal((2 * c - 2) * TJ + b)
      Exit For
     End If
    Next b
   End If

   If MutPos1 = TJ Then
    For b = MutPos1 - 1 To 1 Step -1
     If Gene(NewP, (2 * c - 2) * TJ + MutPos1) = OldGenVal((2 * c - 2) * TJ + b) Then
      Gene(NewP, (2 * c - 2) * TJ + b) = OldGenVal((2 * c - 2) * TJ + MutPos1)
      Gene(NewP, (2 * c - 2) * TJ + MutPos1) = OldGenVal((2 * c - 2) * TJ + b)
      Exit For
     End If
    Next b
   End If

   If MutPos1 > 1 And MutPos1 < TJ Then
    'Search backwards to find a gene with the new generated value
    'swift the gene value if found,otherwise search forwards to find one to match that new
value
    For b = MutPos1 - 1 To 1 Step -1
     If Gene(NewP, (2 * c - 2) * TJ + MutPos1) = OldGenVal((2 * c - 2) * TJ + b) Then
      Gene(NewP, (2 * c - 2) * TJ + b) = OldGenVal((2 * c - 2) * TJ + MutPos1)
      Gene(NewP, (2 * c - 2) * TJ + MutPos1) = OldGenVal((2 * c - 2) * TJ + b)
     End If
    Next b
    'search forwards
    For b = MutPos1 + 1 To TJ
     If Gene(NewP, (2 * c - 2) * TJ + MutPos1) = OldGenVal((2 * c - 2) * TJ + b) Then
      Gene(NewP, (2 * c - 2) * TJ + b) = OldGenVal((2 * c - 2) * TJ + MutPos1)
      Gene(NewP, (2 * c - 2) * TJ + MutPos1) = OldGenVal((2 * c - 2) * TJ + b)
      Exit For
     End If
    Next b
   End If

   'randomly select one mutation position for gene representing job-confiuration selection
   MutPos2 = Int(TJ * Rnd()) + 1
   'radomly genrate a value for the selected gene,subject to the feasibility constraints
   Do
    Gene(NewP, (2 * c - 1) * TJ + MutPos2) = Int(TF(c) * Rnd()) + 1
   Loop Until FEA(c, MutPos2, Gene(NewP, (2 * c - 1) * TJ + MutPos2)) = 1
  Next c
```

```
 'Calculate the fitness value after mutation
   sngFitness(NewP) = sngObjective(NewP)

  If sngFitness(NewP) > sngFitNew Then
    sngFitness(NewP) = sngFitNew
    For c = 0 To 2 * TC - 1
      For j = 1 To TJ
       Gene(NewP, c * TJ + j) = OldGenVal(c * TJ + j)
      Next j
    Next c
  End If
 Next k
End Sub

Function sngObjective(kk As Integer) As Single

Dim GenPos As Integer, GenPosOne As Integer, GenPosTwo As Integer, FirstConf As
Integer, SecondConf As Integer
Dim c As Integer, i As Integer, j As Integer, g As Integer
Dim sngMIC As Single, sngMHC As Single, sngWIP As Single
Dim sngCFC As Single, sngCompTime() As Single, sngBuffTime() As Single,
sngTempTimeSet() As Single
Dim sngTotalDelt As Single, sngPrecedeJobs As Single, sngProcessTime() As Single
Dim clnFinGenPos As New Collection, clnFinGenCon As New Collection

ReDim sngCompTime(0 To TC, 0 To TJ), sngBuffTime(0 To TC, 0 To TJ)
ReDim sngDeltTime(0 To TC, 0 To TJ), sngProcessTime(0 To TC, 0 To TJ, 0 To TJ)

sngMIC = 0
sngMHC = 0
sngCFC = 0

For c = 1 To TC
  For j = 1 To TJ
    'Idle time cost of machine tools
    sngMIC = sngMIC + sngUI * Q(c, j) * sngUMI(c, j, Gene(kk, (2 * c - 1) * TJ + j)) *
sngUPT(c, j, Gene(kk, (2 * c - 1) * TJ + j))
    'Material handling cost
    sngMHC = sngMHC + sngUM * Q(c, j) * sngUMH(c, j, Gene(kk, (2 * c - 1) * TJ + j))
  Next j
Next c

'System reconfiguration costs
For c = 1 To TC
  For GenPosOne = 1 To TJ - 1
```

```
    For j = 1 To TJ
      If Gene(kk, (2 * c - 2) * TJ + j) = GenPosOne Then
        FirstConf = Gene(kk, (2 * c - 1) * TJ + j)
        Exit For
      End If
    Next j
    GenPosTwo = GenPosOne + 1
    For j = 1 To TJ
      If Gene(kk, (2 * c - 2) * TJ + j) = GenPosTwo Then
        SecondConf = Gene(kk, (2 * c - 1) * TJ + j)
        Exit For
      End If
    Next j
    sngCFC = sngCFC + FFC(c, FirstConf, SecondConf)
  Next GenPosOne
Next c

'WIP holding cost
sngWIP = 0
sngTotalDelt = 0
'calculate the processing time of each job in each cell
For c = 1 To TC
  For GenPos = 1 To TJ
    For j = 1 To TJ
      If Gene(kk, (2 * c - 2) * TJ + j) = GenPos Then
        sngProcessTime(c, j, GenPos) = Q(c, j) * sngUPT(c, j, Gene(kk, (2 * c - 1) * TJ +
j))
        Exit For
      End If
    Next j
  Next GenPos
 Next c
 'if a job is assigned in the first position in the sequence
 'the completion time of that job is the processing time of that job
 For c = 1 To TC
    For j = 1 To TJ
      If Gene(kk, (2 * c - 2) * TJ + j) = 1 Then
        sngCompTime(c, j) = sngProcessTime(c, j, 1)
        Exit For
      End If
    Next j
 Next c
 'the completion time of jobs that are not in the first position in the sequence
 For c = 1 To TC
  For GenPos = 2 To TJ
    For j = 1 To TJ
```

```
      If Gene(kk, (2 * c - 2) * TJ + j) = GenPos Then
        'calculate the completion time of jobs that precedes job in position "Genpos".
        sngPrecedeJobs = 0
        For g = 1 To TJ
          For i = 1 To GenPos - 1
            sngPrecedeJobs = sngPrecedeJobs + sngProcessTime(c, g, i)
          Next i
        Next g
        'calculate the completion time of job in position "Genpos"
        sngCompTime(c, j) = sngPrecedeJobs + sngProcessTime(c, j, GenPos)
        Exit For
      End If
    Next j
  Next GenPos
Next c

'buffer the completion time
For c = 1 To TC
  For j = 1 To TJ
    sngBuffTime(c, j) = sngCompTime(c, j)
  Next j
Next c
'Find out the maximun completion time for each product
'and its value is stored in sngBuffTime(1, j)
For j = 1 To TJ
  For c = 2 To TC
    If sngBuffTime(c, j) > sngBuffTime(1, j) Then
      sngBuffTime(1, j) = sngBuffTime(c, j)
    End If
  Next c
Next j
'calculating the difference between the completion time of each job
'and that of the last job for each produbct variant
For j = 1 To TJ
  For c = 1 To TC
    sngDeltTime(c, j) = sngBuffTime(1, j) - sngCompTime(c, j)
    sngTotalDelt = sngTotalDelt + sngDeltTime(c, j)
  Next c
Next j
'calculating the WIP holding cost
sngWIP = sngUW * sngTotalDelt
'Objective value
sngObjective = sngMIC + sngMHC + sngCFC + sngWIP
End Function
```

```
'Progam Name: Solving the Combined Modular Product Scheduling and Production Cell
'             Reconfiguration Problem: a GA Approach with Serial Chromosome
'             Coding
'Programmer:  Hegui Ye
'Date:        February 08,2005
'Decription:  This form is mainly for inputting product and manufacturing systems
'             data and input methods selection
'Folder:      "C:\Documents and Settings\HEGUI YE\My Documents\GA2\

Option Explicit
Dim TM As Integer, TV As Integer, MaxTI As Integer, TI() As Integer
Dim m As Integer, v As Integer, c As Integer, j As Integer, f As Integer
Dim strMsg As String, Demand() As Integer, varResponse As Variant

Private Sub Form_Load()
Load frmInput
frmInput.Show
End Sub


Private Sub txtTotalNumOfProducts_Change()
Dim MaxTV As Integer, InitialTV As Variant, blnFlagProduct As Boolean
TV = CInt(txtTotalNumOfProducts.Text)
TJ = TV
End Sub

Private Sub txtTotalNumOfModules_Change()
Dim blnFlagModule As Boolean, InitialTM As Variant, MaxTM As Integer
TM = CInt(txtTotalNumOfModules.Text)
TC = TM
End Sub

Private Sub txtMaxNumOfInstances_Change()
Dim MaxTI As Integer, InitialTI As Variant, blnFlagInst As Boolean
MTI = CInt(txtMaxNumOfInstances.Text)
End Sub

Private Sub cmdContinue1_Click()
frmSelectionMethod.Enabled = True
chkInputData1.Enabled = True
chkInputData2.Enabled = True
End Sub
Private Sub chkInputData1_Click()

frmManualSelection.Enabled = True
lblProductNumber.Enabled = True
```

```
lblProductQuantity.Enabled = True
txtProductQuantity.Enabled = True
lblModuleNumber.Enabled = True
lblTotalInstanceForModule.Enabled = True
txtTotalInstanceForModule.Enabled = True
lblInstanceNumber.Enabled = True
Combo1.Enabled = True
Call Manual_Selection
End Sub

Private Sub chkInputData2_Click()
Call GA_Selection
Load frmGAInformation
frmGAInformation.Show
Unload frmInput
frmInput.Hide
End Sub
Private Sub GA_Selection()
Open "C:\Documents and Settings\HEGUI YE\My Documents\GA2\Data2-GA.txt" For
Output As #10

Dim MaxDemand As Integer, MinDemand As Integer, MaxFFC As Integer, MinFFC As
Integer
Dim MaxUMH As Integer, MinUMH As Integer, MaxUMI As Integer, MinUMI As
Integer
Dim MaxUPT As Integer, MinUPT As Integer
Dim f1 As Integer, f2 As Integer, j1 As Integer, j2 As Integer, i As Integer
Dim ModuleFEA() As Integer, InstSelection() As Integer, Demand() As Integer
Dim sngMFEA As Single

ReDim TI(TM), TF(TM), Q(TC, TJ), Demand(TV)
ReDim InstSelection(TM, TV), FFC(TC, MTI, MTI), ModuleFEA(TM, MTI, MTI)
ReDim sngUMH(TC, TJ, MTI), sngUMI(TC, TJ, MTI), sngUPT(TC, TJ, MTI),
FEA(TC, TJ, MTI)
MinDemand = 200
MaxDemand = 1000
MinFFC = 20
MaxFFC = 100
MaxUMH = 50
MinUMH = 2
MaxUMI = 20
MinUMI = 1
MaxUPT = 50
MinUPT = 5
'randomly generate an integer between 1 and MTL
'for the total instance number in each module
```

```
Write #10,
Write #10, "Number of instances in each module"
Write #10,
For m = 1 To TM
 TI(m) = Int(Rnd() * MTI) + 1
 TF(m) = TI(m)
 'MsgBox "TI(" & m & ") = " & TI(m)& "TF(" & m & ") = " & TF(m)
Next m
For m = 1 To TM
 Write #10, "TI( " & m & " ) = " & TI(m)
Next m
For m = 1 To TM
 Write #10, "TF( " & m & " ) = " & TF(m)
Next m

'randomly generate an integer between MinDemand and MaxDemand
'for each product variant
Write #10,
Write #10, "Product demand"
For v = 1 To TV
 Demand(v) = Int(Rnd() * (MaxDemand - MinDemand)) + MinDemand
 'MsgBox "Demand(" & v & ") = " & Demand(v)
 Write #10, "Demand( " & v & " ) = " & Demand(v)
Next v
'Calculating the production volume for each job in each cell
Write #10,
Write #10, "Job volume in each cell"
For c = 1 To TC
 For j = 1 To TJ
  Q(c, j) = Demand(j)
  'MsgBox "Job volume Q(" & c & "," & j & ") = " & Q(c, j)
  Write #10, "Q( " & c & "," & j & " ) = " & Q(c, j)
 Next j
 Write #10,
Next c
'randomly select an instance from each module for each product variant
Write #10,
Write #10, "Instance selection for product variants"
For m = 1 To TM
 For v = 1 To TV
    InstSelection(m, v) = Int(Rnd() * TI(m)) + 1
    'MsgBox "InstSelection(" & m & "," & v & ") = " & InstSelection(m, v)
    Write #10, "InstSelection( " & m & "," & v & " ) = " & InstSelection(m, v)
 Next v
 Write #10,
Next m
```

```
'Module-Configuration feasibility matrix
Write #10,
Write #10, "Module-Configuration feasibility matrix"
For m = 1 To TM
 For i = 1 To TI(m)
  For f = 1 To TF(m)
   If f = i Then
    ModuleFEA(m, i, f) = 1 'Guarantee one job has at least its own feasible conf
   Else
    sngMFEA = Round(Rnd(), 2)
    If sngMFEA >= 0.8 Or sngMFEA <= 0.2 Then
     ModuleFEA(m, i, f) = 1
    Else
     ModuleFEA(m, i, f) = 0
    End If
   End If
   'MsgBox "ModuleFEA(" & m & "," & i & "," & f & ") = " & ModuleFEA(m, i, f)
   Write #10, "ModuleFEA( " & m & "," & i & "," & f & " ) = " & ModuleFEA(m, i, f)
  Next f
 Next i
 Write #10,
Next m
'Job-Configuration feasibility matrix is derived from
'the module-Configuration feasibility matrix
Write #10,
Write #10, "Job-Configuration feasibility matrix"
For c = 1 To TC
 For f = 1 To TF(c)
  For j = 1 To TJ
   For i = 1 To TI(c)
    If InstSelection(c, j) = i Then
     FEA(c, j, f) = ModuleFEA(c, i, f)
     'MsgBox "FEA(" & c & "," & j & "," & f & ") = " & FEA(c, j, f)
     Write #10, "FEA( " & c & "," & j & "," & f & " ) = " & FEA(c, j, f)
     Exit For
    End If
   Next i
  Next j
 Next f
 Write #10,
Next c
'Configuration-Configuration cost matrix
Write #10,
Write #10, "Configuration-Configuration cost matrix"
For c = 1 To TC
```

```
 For f1 = 1 To TF(c)
  For f2 = 1 To TF(c)
   If f2 = f1 Then
    FFC(c, f1, f2) = 0
   Else
    FFC(c, f1, f2) = Fix(Rnd() * (MaxFFC - MinFFC) + MinFFC)
   End If
   'MsgBox "FFC(" & c & "," & f1 & "," & f2 & ") = " & FFC(c, f1, f2)
   Write #10, "FFC( " & c & "," & f1 & "," & f2 & " ) = " & FFC(c, f1, f2)
  Next f2
 Next f1
 Write #10,
Next c
'Number of units of material handling cost matrix
Write #10,
Write #10, "Number of units of material handling cost matrix"
For c = 1 To TC
 For j = 1 To TJ
  For f = 1 To TF(c)
   If FEA(c, j, f) = 1 Then
    sngUMH(c, j, f) = Fix(Rnd() * (MaxUMH - MinUMH) + MinUMH)
   Else
    sngUMH(c, j, f) = BigM
   End If
   'MsgBox "sngUMH(" & c & "," & j & "," & f & ") = " & sngUMH(c, j, f)
   Write #10, "sngUMH( " & c & "," & j & "," & f & " ) = " & sngUMH(c, j, f)
  Next f
 Next j
 Write #10,
Next c
'Number of units of machine idle running cost matrix
Write #10,
Write #10, "Number of units of machine idle running cost matrix"
For c = 1 To TC
 For j = 1 To TJ
  For f = 1 To TF(c)
   If FEA(c, j, f) = 1 Then
    sngUMI(c, j, f) = Fix(Rnd() * (MaxUMI - MinUMI) + MinUMI)
   Else
    sngUMI(c, j, f) = BigM
   End If
   'MsgBox "sngUMI(" & c & "," & j & "," & f & ") = " & sngUMI(c, j, f)
   Write #10, " sngUMI( " & c & "," & j & "," & f & " ) = " & sngUMI(c, j, f)
  Next f
 Next j
 Write #10,
```

```
Next c
'unit processing time matrix
Write #10,
Write #10, "unit processing time matrix"
For c = 1 To TC
 For j = 1 To TJ
  For f = 1 To TF(c)
   If FEA(c, j, f) = 1 Then
    sngUPT(c, j, f) = Fix(Rnd() * (MaxUPT - MinUPT) + MinUPT)
   Else
    sngUPT(c, j, f) = BigM
   End If
   'MsgBox "sngUPT(" & c & "," & j & "," & f & ") = " & sngUPT(c, j, f)
   Write #10, "sngUPT( " & c & "," & j & "," & f & " ) = " & sngUPT(c, j, f)
  Next f
 Next j
 Write #10,
Next c
End Sub
Private Sub Manual_Selection()

If TJ = 4 And TC = 5 Then
 Open "C:\Documents and Settings\HEGUI YE\My Documents\GA2\data2-V4C5.txt"
For Input As #2
End If
If TJ = 4 And TC = 4 Then
 Open "C:\Documents and Settings\HEGUI YE\My Documents\GA2\data2-V4C4.txt"
For Input As #2
End If
If TJ = 4 And TC = 3 Then
 Open "C:\Documents and Settings\HEGUI YE\My Documents\GA2\data2-V4C3.txt"
For Input As #2
End If
If TJ = 3 And TC = 4 Then
 Open "C:\Documents and Settings\HEGUI YE\My Documents\GA2\data2-V3C4.txt"
For Input As #2
End If
MTI = TV
 'CC = CELLS; JJ = JOB; FF = CONFIGURATIONS
 Dim Demand() As Integer, CC As Integer, JJ As Integer, FF As Integer
 Dim strMsg As String, strTitle As String

 ReDim Demand(TJ), Q(TC, TJ)
 ReDim FEA(TC, TJ, MTI), FFC(TC, MTI, MTI)
 ReDim sngUMH(TC, TJ, MTI), sngUMI(TC, TJ, MTI)
 ReDim sngUPT(TC, TJ, MTI)
```

```
ReDim TF(TC)

For CC = 1 To TC
 TF(CC) = MTI
Next CC

'Product Demand
For JJ = 1 To TJ
   Input #2, Demand(JJ)
   'MsgBox "Demand(" & JJ & ") = " & Demand(JJ)
Next JJ
'Calculatitng the job volume in each cell
For CC = 1 To TC
 For JJ = 1 To TJ
   Q(CC, JJ) = Demand(JJ)
   'MsgBox "Job volume Q(" & CC & "," & JJ & ") = " & Q(CC, JJ)
 Next JJ
Next CC
'Configuration-Module instance feasibility matrix in each cell
For CC = 1 To TC
 For JJ = 1 To TJ
   For FF = 1 To MTI
    Input #2, FEA(CC, JJ, FF)
    'MsgBox "FEA(" & CC & "," & JJ & "," & FF & ") = " & FEA(CC, JJ, FF)
   Next FF
 Next JJ
Next CC
'Configuration-configuration cost in each cell
For CC = 1 To TC
 For JJ = 1 To TJ
   For FF = 1 To MTI
    Input #2, FFC(CC, JJ, FF)
    'MsgBox "FFC(" & CC & "," & JJ & "," & FF & ") = " & FFC(CC, JJ, FF)
   Next FF
 Next JJ
Next CC
'Number of units of material handling if configuration F is used for job J in cell C
For CC = 1 To TC
 For JJ = 1 To TJ
   For FF = 1 To MTI
    Input #2, sngUMH(CC, JJ, FF)
    'MsgBox "sngUMH(" & CC & "," & JJ & "," & FF & ") = " & sngUMH(CC, JJ, FF)
   Next FF
 Next JJ
Next CC
'Number of units of idl-running machine if configuration F is used for job J in cell C
```

```
For CC = 1 To TC
  For JJ = 1 To TJ
    For FF = 1 To MTI
      Input #2, sngUMI(CC, JJ, FF)
      'MsgBox "sngUMI(" & CC & "," & JJ & "," & FF & ") = " & sngUMI(CC, JJ, FF)
    Next FF
  Next JJ
Next CC
'Number of units of WIP if configuration F is used for job J in cell C
For CC = 1 To TC
  For JJ = 1 To TJ
    For FF = 1 To MTI
      Input #2, sngUPT(CC, JJ, FF)
      'MsgBox "sngUPT(" & CC & "," & JJ & "," & FF & ") = " & sngUPT(CC, JJ, FF)
    Next FF
  Next JJ
Next CC

Close #2

strMsg = "Data are all loaded. Choose <OK> to display GA information."
strTitle = "Data Loaded"
MsgBox strMsg, vbOKOnly & vbExclamation, strTitle
If vbOK Then
Load frmGAInformation
frmGAInformation.Show
End If

End Sub
Private Sub txtProductQuantity_Change()
Dim v As Integer, MaxDemand As Integer
Dim InitialDemand As Variant, blnFlagDemand As Boolean
Dim varInstSelection As Boolean, varInstNumber As Variant

lstTotalInstance.Enabled = True
lstTotalInstance.Text.Enabled = True
For i = 1 To MTI
lstTotalInstance.AddItem i
Next i
For m = 1 To TM
  lstTotalInstance = m
  TI(m) = lstTotalInstance.Selected
  TF(m) = TI(m)
Next m

ReDim Demand(TV)
```

```
MaxDemand = 10000
For v = 1 To TV
 txtProductNumber.Text = v

 Demand(v) = txtProductQuantity.Text

Next v
End Sub

 If TM <= 5 Then
  strMsg = "Choose <Yes> to manually determine the number of instances for each
module or " & Chr(13) & _
         "Select <No> to let the GA randomly generate the number of instances for each
module."
  varInstNumber = MsgBox(Prompt:=strMsg, _
                  Buttons:=vbYesNo + vbInformation + vbDefaultButton2, _
                  Title:="Instance Selection")
  If varInstNumber = vbYes Then
    Call Manual_Selection
  End If
Else
Call GA_Selection
End If
```

```
'Progam Name: Solving the Combined Modular Product Scheduling and Production Cell
'              Reconfiguration Problem: a GA Approach with Serial Chromosome
'              Coding
'Programmer:   Hegui Ye
'Date:         February 08,2005
'Decription:   This form is used to select the GA parameter
'Folder:       "C:\Documents and Settings\HEGUI YE\My Documents\GA2\

Option Explicit

Private Sub chkCrossoverOption_Click(Index As Integer)
'one-one cutting
If chkCrossoverOption(0) = True Then
  CrossRule = 1
End If
'one-two cutting
If chkCrossoverOption(1) = True Then
  CrossRule = 2
End If
'two-one cutting
If chkCrossoverOption(2) = True Then
  CrossRule = 3
End If
'two-two cutting
If chkCrossoverOption(3) = True Then
  CrossRule = 4
End If
End Sub

Private Sub chkMutationOption_Click(Index As Integer)
If chkMutationOption(0) = True Then
  sngMutRate = 0.01
End If
If chkMutationOption(1) = True Then
  sngMutRate = 0.02
End If
If chkMutationOption(2) = True Then
  sngMutRate = 0.05
End If
If chkMutationOption(3) = True Then
  sngMutRate = 0.1
End If
End Sub

Private Sub chkParentSelection_Click(Index As Integer)
```

```
If chkParentSelection(0) = True Then
  ParentChoice = 1
End If
If chkParentSelection(1) = True Then
  ParentChoice = 2
End If
End Sub

Private Sub chkPopOption_Click(Index As Integer)
If chkPopOption(0) = True Then
  PopSize = 4 * (TC + TJ)
End If
If chkPopOption(1) = True Then
  PopSize = 6 * (TC + TJ)
End If
If chkPopOption(2) = True Then
  PopSize = 8 * (TC + TJ)
End If
End Sub

Private Sub chkShowResults_Click()
Load frmFinalResults
frmFinalResults.Show
frmFinalResults.Refresh
End Sub

Private Sub cmdEnd_Click()
End
End Sub

Private Sub cmdRunAll_Click()
Dim Pa As Integer, Po As Integer, Mu As Integer, Cr As Integer
Dim sngTemp As Single, sngCurrentObject() As Single
ReDim sngCurrentObject(2, 3, 4, 4)

Open "C:\Documents and Settings\HEGUI YE\My Documents\GA2\Data1-Runall.txt"
For Output As #11
lngStartTime = 0
lngElapseTime = 0
lngStartTime = Hour(Time) * 3600 + Minute(Time) * 60 + Second(Time)
For Pa = 1 To 2
  ParentChoice = Pa
  For Po = 1 To 3
    If Po = 1 Then
      PopSize = 4 * (TC + TJ)
    End If
```

```
    If Po = 2 Then
      PopSize = 6 * (TC + TJ)
    End If
    If Po = 3 Then
      PopSize = 8 * (TC + TJ)
    End If
    For Mu = 1 To 4
      If Mu = 1 Then
        sngMutRate = 0.01
      End If
      If Mu = 2 Then
        sngMutRate = 0.02
      End If
      If Mu = 3 Then
        sngMutRate = 0.05
      End If
      If Mu = 4 Then
        sngMutRate = 0.1
      End If
      For Cr = 1 To 4
        If Cr = 1 Then
          CrossRule = 1
        End If
        If Cr = 2 Then
          CrossRule = 2
        End If
        If Cr = 3 Then
          CrossRule = 3
        End If
        If Cr = 4 Then
          CrossRule = 4
        End If
        'call sub GA to run
        Call GA
        Write #11, PopSize, sngMutRate, ParentChoice, CrossRule
        Write #11, sngBestObject
        Write #11,
        sngCurrentObject(Pa, Po, Mu, Cr) = sngBestObject
      Next Cr
    Next Mu
  Next Po
Next Pa
'calculate the minimum value among sngCurrentObject(Pa, Po, Cr, Mu)
For Pa = 1 To 2
  For Po = 1 To 3
    For Mu = 1 To 4
```

```
    For Cr = 2 To 4
      If sngCurrentObject(Pa, Po, Mu, Cr) < sngCurrentObject(1, 1, 1, 1) Then
        sngTemp = sngCurrentObject(Pa, Po, Mu, Cr)
        sngCurrentObject(Pa, Po, Mu, Cr) = sngCurrentObject(1, 1, 1, 1)
        sngCurrentObject(1, 1, 1, 1) = sngTemp
      End If
    Next Cr
  Next Mu
 Next Po
Next Pa

'Calculate elapsed time GA has used
 lngElapseTime = Hour(Time) * 3600 + Minute(Time) * 60 + Second(Time) -
lngStartTime
 'if the program runs overnight
 If lngElapseTime < 0 Then
  lngElapseTime = (Hour(Time) + 24) * 3600 + Minute(Time) * 60 + Second(Time) -
lngStartTime
 End If

Load frmFinalResults
frmFinalResults.Show
With frmFinalResults
    .lblValueOfMaxIter.Caption = lngMaxIter
    .lblValueOfIterate.Caption = lngNumIter
    .lblValueOfMaxTime.Caption = lngMaxRunTime
    .lblValueOfRunTime.Caption = lngElapseTime
    .lblValueOfOptimum.Caption = Round(sngFitness(1), 3)
    .lblValueOfCurrent.Caption = Round(sngCurrentObject(1, 1, 1, 1), 3)
End With

frmFinalResults.Refresh
Close #11
End Sub

Private Sub cmdRunGA_Click()

lngStartTime = 0
lngElapseTime = 0
lngStartTime = Hour(Time) * 3600 + Minute(Time) * 60 + Second(Time)

Call GA

'Calculate elapsed time GA has used
lngElapseTime = Hour(Time) * 3600 + Minute(Time) * 60 + Second(Time) -
lngStartTime
```

```
Load frmFinalResults
frmFinalResults.Show
With frmFinalResults
    .lblValueOfMaxIter.Caption = lngMaxIter
    .lblValueOfIterate.Caption = lngNumIter
    .lblValueOfMaxTime.Caption = lngMaxRunTime
    .lblValueOfRunTime.Caption = lngElapseTime
    .lblValueOfOptimum.Caption = Round(sngFitness(1), 3)
    .lblValueOfCurrent.Caption = Round(sngBestObject, 3)
End With

frmFinalResults.Refresh
End Sub

Private Sub Form_Load()

lngMaxIter = 6 * (TC + TJ + MTI)
lngMaxRunTime = 600

txtMaxRunTime.Text = lngMaxRunTime
txtMaxNumIteration.Text = lngMaxIter
End Sub
```

```
'Progam Name: Solving the Combined Modular Product Scheduling and Production Cell
'                Reconfiguration Problem: a GA Approach with Serial Chromosome
'                Coding
'Programmer:   Hegui Ye
'Date:          February 08,2005
'Decription:    This form lists the computational results
'Folder:        "C:\Documents and Settings\HEGUI YE\My Documents\GA2\

Private Sub cmdEnd_Click()
   frmFinalResults.Hide
   Unload frmFinalResults
   End
End Sub

Private Sub cmdSave_Click()
  Open "C:\Documents and Settings\HEGUI YE\My Documents\GA2\Data1.txt" For
Output As #1

  Dim strMsg As String, strTitle As String
  Dim blnResponse As Boolean

  Write #1,
  Write #1, "GA for Scheduling the Production of Modular Products in RMS
Environment"
  Write #1,
  Write #1,
  Write #1, "Parent Selection = " & ParentChoice
  Write #1, "Population Size = " & PopSize
  Write #1, "Crossover Rule = " & CrossRule
  Write #1, "Mutation Rate = " & sngMutRate
  Write #1,
  Write #1, "Stopping Critirion-Maximum Iteration = " & lngMaxIter
  Write #1, "Total Iteration Number = " & lngNumIter
  Write #1, "Elapsed Runtime = " & lngElapseTime
  Write #1,
  Write #1, "Objective Value = " & sngBestObject
  Write #1,
  Write #1, "Job-Position assignments"

   For c = 1 To TC
    For j = 1 To TJ
     For pp = 1 To TJ
      If X(c, j, pp) = 1 Then
        Write #1, "X(" & c & "," & j & "," & pp & ")= " & X(c, j, pp)
      Else
        X(c, j, pp) = 0
```

```
      End If
     Next pp
    Next j
    Write #1,
   Next c

  Write #1,
  Write #1, "Job-Configuration selection"
  For c = 1 To TC
   For j = 1 To TJ
    For f = 1 To TF(c)
     If Z(c, j, f) = 1 Then
      Write #1, "Z(" & c & "," & j & "," & f & ")= " & Z(c, j, f)
      'Write #4, "Configuration " & f & "is selected for processing job " & j & "in cell "
& c
      End If
    Next f
   Next j
   Write #1,
  Next c

  Close #1

  strMsg = "Data saved. Choose <End> botton to exit the program?"
  strTitle = "Save data"
  blnResponse = MsgBox(strMsg, vbOKOnly + vbQuestion + vbDefaultButton1, strTitle)

End Sub

Private Sub Form_Load()

  lblValueOfMaxIter.Caption = 0
  lblValueOfIterate.Caption = 0
  lblValueOfMaxTime.Caption = 0
  lblValueOfRunTime.Caption = 0
  lblValueOfOptimum.Caption = 0
  lblValueOfCurrent.Caption = 0
End Sub
```