Genetic algorithm for production scheduling

Mary El-Mallakh

GENETIC ALGORITHM FOR
PRODUCTION SCHEDULING

MARY EL MALLAKH

2000

The American University in Cairo

GENETIC ALGORITHM FOR PRODUCTION SCHEDULING

A Thesis Submitted to

The Computer Science Department

in partial fulfillment of the requirements for

the degree of Master of Science

by

Mary El Mallakh

B.Sc. in Computer Science

under the supervision of Dr. A. Abdelbar

April/2000

2000/45

The American University in Cairo

GENETIC ALGORITHM FOR PRODUCTION SCHEDULING

A Thesis Submitted by Mary El Mallakh

In Department of Computer Science

April/2000

in partial fulfillment of the requirements for

the degree of Master of Science

has been approved by

Dr. Ashraf Abdelbar [redacted]
Thesis Committee Chair / Adviser
Affiliation

Drs. M.N. Mikhail,
Thesis Committee Reader/ Examiners/
Affiliation

Dr. prof / ISMAIL AMR ISMAIL [redacted]
Thesis Committee Reader / Examiner
Affiliation

[redacted] May 31, 2000 [redacted] June 4, 00
Department Chair/ Program Director — Date — Dean — Date

The American University in Cairo

LIBRARY

Theses

Declaration to be signed by the Author

Name: ..... Mary El Mallakh .....

Title of the Thesis: .. Genetic Algorithm for Production Scheduling .....

.....

Department: Computer Science Year: .... 2000 .... (Library No 2000/45 ..)

Please Sign and date ONE of the following paragraphs:

1. The Thesis may be consulted in the Library and Photocopied.

Signed.. [redacted] ..... Date: .. 31 May 2000 ..

OR

2. The Thesis may be consulted in the Library, but may not be photocopied. The restriction on photocopying will cease two years after the date below, unless I apply for, and am granted its renewal. *

Signed: ..... Date: .....

OR

3. The Thesis may neither be consulted nor photocopied without written permission from me or from the appropriate Head of Department if I cannot be contacted. This restriction will cease three years after the date below, unless I apply for, and am granted its renewal. *

Signed: ..... Date: .....

* Application for renewal of restrictions will be considered by the Librarian, the appropriate Head of Department and the Chairman of the Academic Board or his nominee.

ABSTRACT

Planning and scheduling are common to many different engineering domains. Whether the project is as large as the Boston Harbor Tunnel or something as seemingly simple as the redesign of the packaging for a tape dispenser, both planning and scheduling are profoundly important.

Practical scheduling problems are numerous and varied. However, many of them share two important characteristics; they are very difficult, and good quality solutions bring highly tangible benefits. In general, scheduling problems are NP-hard, consequently there are no known algorithms guaranteed to give an optimal solution and run in polynomial time. [Husbands 1994]

This is not a new subject. Planning and scheduling methods have been proposed and analyzed since at least the 1950s. Although methods exist for finding optimal solutions to some specific scheduling problem formulations, many methods do not work when the structures of the constraints or objectives change. In addition, many methods do not perform well when faced with problems of significant size. In many cases, simply finding feasible solutions is a considerable challenge. The difficult nature of resource-constrained scheduling led Tavares and Weglarz [Tavares and Weglarz 1990] to label project management and scheduling "a permanent challenge for operations research".

The complex, combinatorial nature of most scheduling problems has led many researchers to experiment with genetic algorithms as a solution method. Commonly touted for their ability to solve nonlinear and combinatorial problems, genetic algorithms typically perform well on problems in which the objective and/or search space combine both discrete and continuous variables. They are also often noted for

searching large, multi-modal spaces effectively since they operate on a population of solutions rather than on one individual and use no gradient or other problem-specific information.

"This thesis describes the production planning problem of a printed circuit board assembly plant. The problem is a generalization of the so-called flexible flow line scheduling problem. The process engineer recognizes it as a dynamic sequence of updates in the production plan." [Johnsson et al.1997] An algorithm for PCBs assembly line balancing (GAPS) is presented. Whereas traditional scheduling methods use search or scheduling rules (heuristics), this method uses a genetic algorithm. This solution method introduces a sequence-based representation that encodes the schedule information as a triple array of component types, number of component types and machines.

"*It is ironic, but perhaps not surprising, that our attempts to improve simulated evolution as an optimization procedure continue to take us closer to real biological systems.*" [Hillis 1990]

"*Project management and scheduling is a permanent challenge for operations research*". [Tavares and Weglarz 1990]

ACKNOWLEDGEMENTS

Through out my work on this thesis a number of individuals have shaped my ideas on Genetic Algorithms and Printed Circuit Board production. For their contributions, I especially thank Dr. Ahmed Sameh, Dr. Grant Boctor and Michael Hanke.

A special acknowledgment goes to Dr. Abdelbar for his continuous support. I would like to thank Dr. Mikhail and everyone in the Computer Science Department.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I INTRODUCTION

Planning and scheduling are common to many different engineering domains. Whether the project is as large as the Boston Harbor Tunnel or something as seemingly simple as the redesign of the packaging for a tape dispenser, both planning and scheduling are profoundly important. Even on a small project, the number of possible courses of action and the number of ways to allocate resources quickly become overwhelming. On a factory floor, determining which jobs should be executed on which machines by which employees can mean the difference between significant profit and debilitating loss. In a software development shop, assigning responsibility for tasks and effectively managing disruptions can mean the difference between a product that ships in time to hit a market window and a product that misses that window.

Practical scheduling problems are numerous and varied. However, many of them share two important characteristics; they are very difficult, and good quality solutions bring highly tangible benefits. In general, scheduling problems are NP-hard, consequently there are no known algorithms guaranteed to give an optimal solution and run in polynomial time. [Husbands 1994]

In its most general form, the resource-constrained scheduling problem asks the following: Given a set of activities, a set of resources, and a measurement of performance, what is the best way to assign the resources to the activities such that the performance is maximized? The general problem encapsulates many variations such

as the job-shop and flow-shop problems, production scheduling, and the resource-constrained project-scheduling problem.

Scheduling requires the integration of many different kinds of data. Constructing a schedule requires models of processes, definition of relationships between tasks and resources, definition of objectives and performance measures, and the underlying data structures and algorithms that tie them all together. Schedules assign resources to tasks (or tasks to resources) at specific times. Tasks (activities) may be anything from machining operations to development of software modules. Resources include people, machines, and raw materials. Typical objectives include minimizing the duration of the project, maximizing the net present value of the project, or minimizing the number of products that are delivered late.

Planning and scheduling are distinctly different activities. The plan defines what must be done and restrictions on how to do it; the schedule specifies both how and when it will be done. The plan refers to the estimates of time and resource for each activity, as well as the precedence relationships between activities and other constraints. The schedule refers to the temporal assignments of tasks and activities required for actual execution of the plan. In addition, any project includes a set of objectives used to measure the performance of the schedule and/or the feasibility of the plan. The objectives determine the overall performance of the plan and schedule.

Although often treated separately, planning and scheduling are often inseparably connected. Changes to a schedule may require a different set of activities in order to produce a feasible schedule. Conversely, a plan may have no corresponding feasible schedule. In either case, objectives such as "minimize makespan", while independent of the plan or schedule, determine the value of a plan

and schedule. Both the plan and the objectives determine the difficulty of finding a schedule.

Scheduling problems are dynamic and are based on incomplete data. No schedule is static until the project is completed, and most plans change almost as soon as they are announced. Depending on the duration of the project, the same may also be true for the objectives. The dynamics may be due to poor estimates, incomplete data, or unanticipated disturbances. As a result, finding an optimal schedule is often confounded not only by meeting existing constraints but also adapting to additional constraints and changes to the problem structure.

Scheduling problems include many types of constraints. Constraints appear in many forms:

- temporal constraints such as "James can work only on Tuesdays, Thursdays, and Fridays";
- precedence constraints such as "The design of the interface can be started when library programming interface is frozen and the analog-to-digital hardware is 75% completed";
- availability constraints such as "three skilled machinists are available in the second shift, four are available in the third shift"; and
- combinations such as "The injection molding machine can run three shifts between maintenance cycles".

Constraints turn a relatively smooth solution space with many optima to a very non-uniform space with few feasible solutions. A typical plan includes many

bottlenecks with little flexibility for change, as well as parts that are almost unconstrained. [Wall 1996]

This is not a new subject. Planning and scheduling methods have been proposed and analyzed since at least the 1950s. Although methods exist for finding optimal solutions to some specific scheduling problem formulations, many methods do not work when the structures of the constraints or objectives change. In addition, many methods do not perform well when faced with problems of significant size. In many cases, simply finding feasible solutions is a considerable challenge. The difficult nature of resource-constrained scheduling led Tavares and Weglarz [Tavares and Weglarz 1990] to label project management and scheduling "a permanent challenge for operations research".

In general, scheduling problems are NP-hard. As a result, most research has been devoted to either simplifying the scheduling problem to the point where some algorithms can find solutions, or to devising efficient heuristics for finding good solutions. Many solution methods have been proposed and implemented. Early approaches solved simplified versions of the problem exactly, but researchers quickly realized that real problems are too large and complicated for any exact solution. For example, decision trees were used to enumerate every possible choice. Heuristic methods were then devised to find good solutions, or to find simply feasible solutions for the really difficult problems. Most research now consists of designing better heuristics for specific instances of scheduling problems. However, heuristic solutions are typically limited to a specific set of constraints or problem formulation, and devising new heuristics is difficult at best.

The complex, combinatorial nature of most scheduling problems has led many researchers to experiment with genetic algorithms as a solution method. Commonly touted for their ability to solve nonlinear and combinatorial problems, genetic algorithms typically perform well on problems in which the objective and/or search space combine both discrete and continuous variables. They are also often noted for searching large, multi-modal spaces effectively since they operate on a population of solutions rather than on one individual and use no gradient or other problem-specific information.

Genetic algorithms are a stochastic search method introduced in the 1970s in the United States by John Holland [Holland 1975] and in Germany by Ingo Rechenberg [Rechenberg 1973]. Based on simplifications of natural evolutionary processes, genetic algorithms operate on a population of solutions rather than a single solution and employ heuristics such as selection, crossover, and mutation to evolve better solutions.

Although genetic algorithms have been studied for over 25 years, implementing them is often as much an art as designing efficient heuristics. Much of the genetic algorithm literature is devoted to relatively simple problems. Simplistic application of a genetic algorithm to small problems often produces reasonable results, but naive application of genetic algorithms to larger problems often results in poor performance. This is due to both the nature of the genetic search and the relationships between a genetic representation and the genetic operators. Direct representation of problems, i.e. use of data types other than bit strings promises further improvements in genetic algorithm applicability, robustness, and performance. Continued reduction in computational cost along with increases in power and speed make genetic algorithms viable alternatives despite their significant computational overhead.

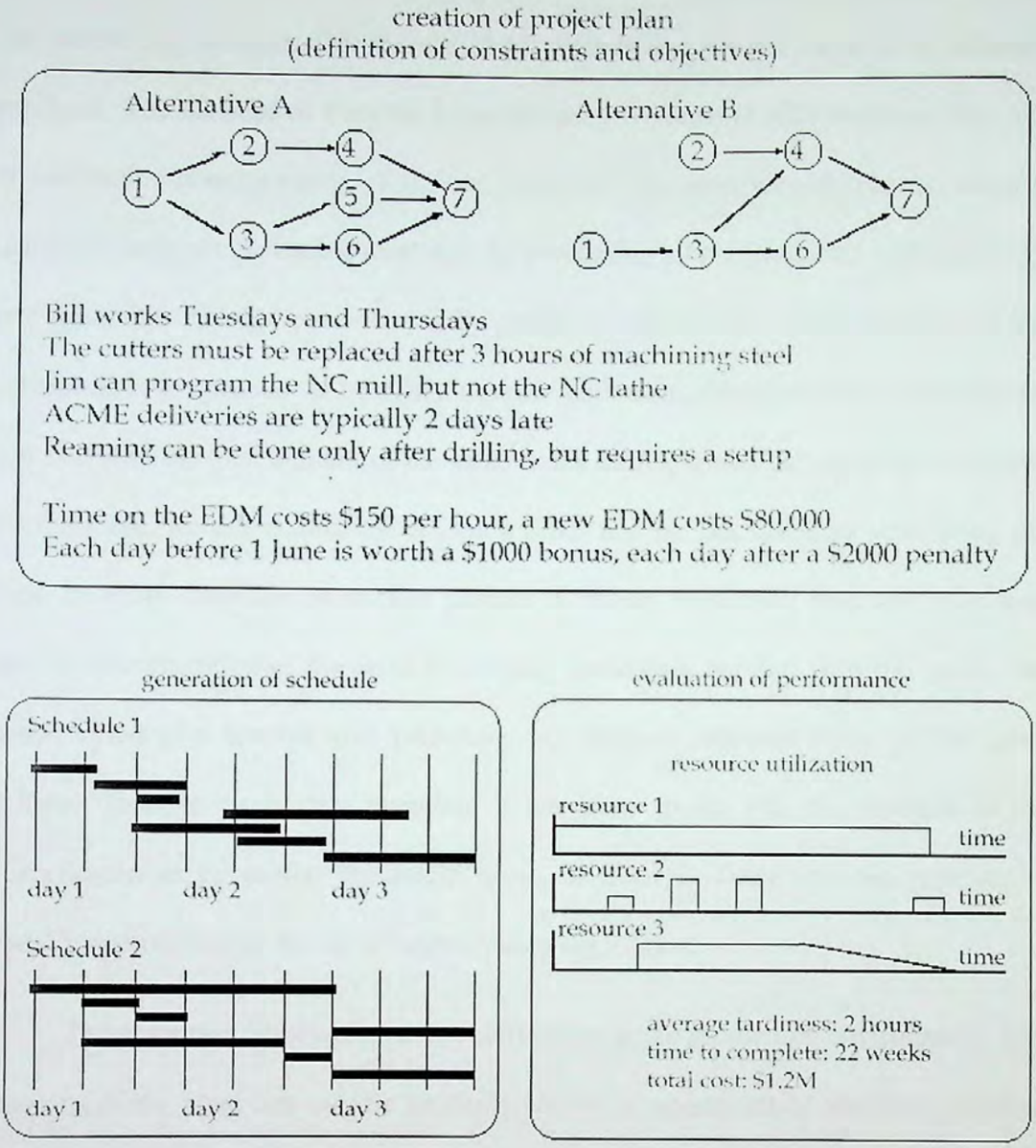

Figure 1: the parts of a resource-constrained scheduling problem. A plan is a set of constraints that define the problem. A schedule consists of a set of assignments of resources to activities at specific times. Resources include people, machines, and raw materials. Constraints define the limits of resources and relations between activities. Objectives define the goals and performance measures for the problem. [Wall 1996]

Due to the continued challenge of resource-constrained scheduling and the promising performance of genetic algorithms on similar problems, scheduling problems have attracted a great deal of attention in the genetic algorithm community in the past five years. However, most implementations are variations of traditional operations research approaches to solving scheduling problems.

It is characteristic of many branches of advanced production technologies that the production program comprises a large number and several variants of different products. The concept of Flexible Manufacturing Systems (FMS) supports easy and cost-effective manufacturing of various products. The main idea of these systems is using the same set of flexible machines for processing all the products. A change from one product to another requires a setup operation, the time of which depends on the current and next product to be manufactured. With a suitable production sequence we can decrease the total amount of the setup times during which the machines stay idle. Furthermore, we can control the finishing times and the due dates by scheduling the jobs. In many cases the production planner is mainly concerned with the due dates, and the minimization of the setup times may sometimes conflict with this goal. The construction of a feasible and, preferably, an efficient schedule is one of the most difficult tasks in production planning. It has been shown that the problem is too complicated to be solved accurately (even in theory). Therefore, the problem is usually approached by the use of approximate algorithms.

There are two features that cause difficulties in the production environment. The manufacturing plant has usually multiple copies of operationally identical machine types organized as machine banks. The banks can also be considered as production phases. Each product has to pass these phases in a preceded order (flow line processing). Hence, the system may process several different types of products at a given time. The routing of the products affects the total production efficiency. The second difficulty originates from the fact that the production plan is subjected to due date constraints which imply the preference of feasible schedules of the jobs (i.e., product batches). Therefore, searching for an optimal schedule is not reasonable objective in applications of practical value.

There are three different approaches to the flow line scheduling. In the algorithmic approach the scheduling task is expressed as a mathematical optimization problem and is solved by an approximate algorithm. In the interactive scheduling the production designer uses computer simulation to evaluate different schedules. The hybrid approach integrates both approaches; the algorithms are used to produce a set of possible schedules, which can be then evaluated and manipulated by the interactive scheduling tool. [Hayrinen et al. 1998]

This thesis presents an algorithm called GAPS for finding near-optimal solutions to Printed Circuit Boards (PCBs) assembly line balancing. Whereas traditional scheduling methods use search or scheduling rules (heuristics), GAPS uses a genetic algorithm.

The second chapter of this thesis presents a description of the resource-constrained scheduling problem and a summary of some of its variations. The third chapter contains a brief overview about genetic algorithms. The fourth chapter presents an overview about PCBs production-scheduling problem with special emphasis on line balancing problem. The fifth chapter is a description of the GAPS solution method as well as the test problem and results. Finally, the sixth chapter offers conclusion and suggestion for future work.

CHAPTER II RESOURCE-CONSTRAINED SCHEDULING PROBLEM

Although related and often tightly coupled, planning and scheduling are distinctly different activities. Planning is the construction of the project/process model and definition of constraints/objectives. Scheduling refers to the assignment of resources to activities (or activities to resources) at specific points in, or duration of, time. The definition of the problem is thus primarily a planning issue, whereas the execution of the plan is a scheduling issue. Yet planning and scheduling are coupled; the performance of the scheduling algorithm depends on the problem formulation, and the problem formulation may benefit from information obtained during scheduling. [Wall 1996]

The Problem General Form

In its most general form, the resource-constrained scheduling problem is defined as follows:

Given

- a set of activities that must be executed,
- a set of resources with which to perform the activities,
- a set of constraints which must be satisfied, and
- a set of objectives with which to judge a schedule's performance,

What is the best way to assign the resources to the activities at specific times such that all of the constraints are satisfied and the best objective measures are produced?

The general form includes the following characteristics:

- each task may be executed in more than one manner, depending on which resource(s) is (are) assigned to it
- task precedence relationships may include overlap so that a given task may begin when its predecessor is partially complete
- each task may be interrupted according to a pre-defined set of interruption modes (specific to each task), or no interruption may be allowed
- each task may require more than one resource of various types
- a task's resource requirements may vary over the duration of the task the resources may be renewable (e.g. labor, machines) or non-renewable (e.g. raw materials)
- resources availability may vary over the duration of the schedule or task
- resources may have temporal restrictions

In order to accurately model the uncertainty common in real problems, the general formulation includes the following dynamic characteristics:

- resource availability may change
- resource requirements may change
- objectives may change

Instances of the generalized problem

As noted by Sprecher, the flow shop, job shop, open shop, and assembly line balancing problems are all instances of the general resource-constrained problem

[Sprecher 1994]. In addition, many production scheduling problems, single-mode resource-constrained project scheduling, multi-mode resource-constrained project scheduling, and multiple-project scheduling are also variations of the general problem.

What makes scheduling problems hard?

Aside from the sheer volume of data and management of information required to schedule a project or machine shop, there are some inherent difficulties to solving even simplified scheduling problems.

- Scaling Issues - The Size of the Problem
- Uncertainty and the Dynamic Nature of Real Problems
- Infeasibility - Sparseness of the Solution Space

Production Scheduling

Scheduling allocates resources over time in order to perform a number of tasks. Typically resources are limited and therefore tasks are assigned to resources in a temporal order. From an economic point of view limited resources are scare goods and consequently the problem of task scheduling is of more than just academic relevance.

Following Van Dyke Parunak (1992) scheduling is circumscribed by asking what has to be done where and when. A task occupies a dedicated resource exclusively for some period of time. A group of task primitives may form a complex, in which several tasks have to pass resources in a certain order. In this way the temporal order of resource allocations is restricted by dependencies among the task primitives.

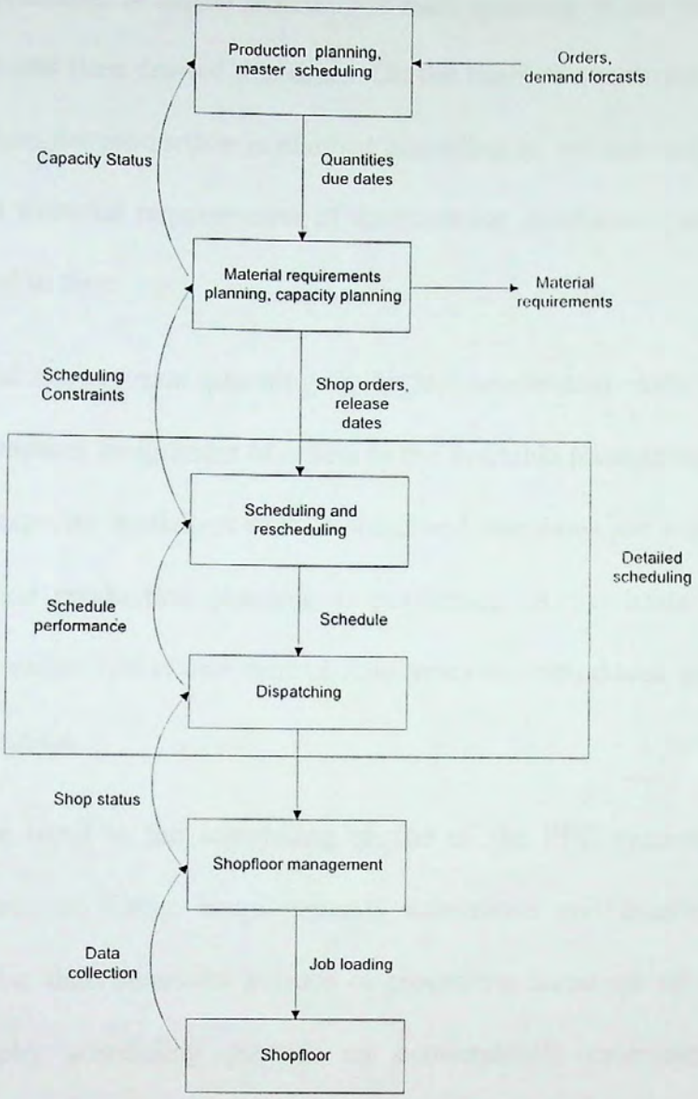

Figure 2: Information flow in a manufacturing system.

In a manufacturing environment, scheduling allocates machines for processing a number of jobs. This function is embedded in the domain of production planning and control (PPC). [Scheer 1989] The purposes covered by a PPC system are outlined best by considering the information flow in a manufacturing system. Figure 2 is taken from Pinedo (1995) and sketches a simplified information flow while neglecting the interfaces to other functions of a manufacturing environment.

Demand forecast and production planning are input to the medium- to long-term production planning. A master schedule is built resulting in the demand of end product quantities and their desired due dates. On the basis of quantities and due date the material required for production is planned according to volume and period. This process results in material requirements of forthcoming production periods, which have to be supplied in time.

The material requirement planning is highly interwoven with the capacity planning. Here, temporal assignment of orders to the available processing capacity are shifted such that capacity bottlenecks are avoided and due dates are kept. Up to this stage coarse-grained production planning is performed on the basis of customer orders. Now shop orders (jobs) and their release times are introduced as an outcome of the capacity planning.

The jobs are input to the scheduling engine of the PPC system. Production scheduling performs lot sizing, keeps capacity constraints and finally produces a detailed schedule i.e. determines the periods of processing some job on its dedicated machines. Thereby scheduling pursues an economically motivated objective. Typically, increasing the throughput of jobs pursues a reduction of the work in-process inventory. Moreover, scheduling aims to avoid delivery delays of customer orders and tries to make full use of the available production capacity. Production planning finishes with dispatching already scheduled jobs to the shop floor management.

In manufacturing systems operations (tasks) are processed by machines (resources) for a certain processing time (time period). Typically, the number of machines available is limited and a machine can process a single operation at a time.

Demand forecast and production planning are input to the medium- to long-term production planning. A master schedule is built resulting in the demand of end product quantities and their desired due dates. On the basis of quantities and due date the material required for production is planned according to volume and period. This process results in material requirements of forthcoming production periods, which have to be supplied in time.

The material requirement planning is highly interwoven with the capacity planning. Here, temporal assignment of orders to the available processing capacity are shifted such that capacity bottlenecks are avoided and due dates are kept. Up to this stage coarse-grained production planning is performed on the basis of customer orders. Now shop orders (jobs) and their release times are introduced as an outcome of the capacity planning.

The jobs are input to the scheduling engine of the PPC system. Production scheduling performs lot sizing, keeps capacity constraints and finally produces a detailed schedule i.e. determines the periods of processing some job on its dedicated machines. Thereby scheduling pursues an economically motivated objective. Typically, increasing the throughput of jobs pursues a reduction of the work in-process inventory. Moreover, scheduling aims to avoid delivery delays of customer orders and tries to make full use of the available production capacity. Production planning finishes with dispatching already scheduled jobs to the shop floor management.

In manufacturing systems operations (tasks) are processed by machines (resources) for a certain processing time (time period). Typically, the number of machines available is limited and a machine can process a single operation at a time.

Often operations cannot be processed in arbitrary orders but obey a prescribed processing order. Jobs often follow technological constraints, which define a certain type of shop floor. In a flow shop all jobs pass the machines in an identical order. In a job shop technological constrains may differ from job to job. In an open job exists no technological restriction and therefore the operations of jobs may be processed in arbitrary orders.

Production scheduling determines starting times of operations without violating technological constraints such that processing times of identical machines do not overlap in time. The resulting timetable is called a schedule. Thereby scheduling pursues at least one economical objective. Typically objectives are the reduction of the makespan of an entire production program, the minimization of the mean job tardiness, the maximization of machine load or some weighted average of many similar criteria.

Job Shop Scheduling

Within the great variety of production scheduling problems the general job shop problem (JSP) is probably the most studied one by academic research during the last decade. It has earned a reputation for being notoriously difficult to solve. It illustrates at least some of the demands required by a wide array of real word problems.

Representation of the JSP

Consider a shop floor where jobs are processed by machines. Each job consists of a certain number of operations. Each operation has to be processed on a dedicated machine and for each operation a processing time is defined. The machine order of operations is prescribed for each job by a technological production recipe. These

technological constraints are therefore static to a problem instance. The basic JSP is a static optimization problem, since all information about the production program is known in advance. Furthermore, the JSP is purely deterministic, since processing times and constraints are fixed and no stochastic events occur.

The most widely used objective is to find a feasible schedule such that the completion time of the total production program (i.e. the makespan) is minimized. Feasible schedules are obtained by permuting the processing order of operations on the machines (operation sequence) but without violating the technological constraints. Accordingly we face a combinatorial minimization problem with constrained permutation of operations. A schedule for a problem instance consists of operations sequences for each machine involved. Since each operation sequence can be permuted independently of the operation sequence of other machines, we have a maximum of $(n!)^m$ different solutions to a problem instance, where n denotes the number of jobs and m denotes the number of machines involved. According to Garey and Johnson (1979) the JSP is an NP-hard problem. The complete restrictions of the basic JSP are: -

- No two operations of one job may be processed simultaneously.
- No preemption of operations is allowed.
- No job is processed twice on the same machine.
- Each job must be processed to completion.
- Jobs may be started at any time, no release times exist.
- Jobs may be finished at any time, no due dates exist.
- Jobs must wait for the next machine to be available.

- No machine may process more than one operation at a time.
- Machine Setup times are negligible.
- There is only one of each type of machine.
- Machines may be idle within the schedule period.
- Machines are available at any time.
- The technological constraints are known in advance and are immutable.

The set of constraints involved in real world applications is much more complex.

A problem instance consists of n jobs and m machines, where J_j denotes the j^{th} job ($1 \leq j \leq n$) and M_i denotes the i^{th} machine ($1 \leq i \leq m$). The machine order (technological constraints) for job J_j is given by $\varphi_j = (M_{\varphi jh})$ ($1 \leq h \leq m$), where h denotes the h^{th} operation of J_j. The processing time of an operation of job J_j to be performed on machine M_i is given by p_{ji}. The technological constraints φ as well as processing times p are given problem data.

The processing order (machine sequence) for machine M_i is given by $\nu_i = (J\nu_{ik})$ ($i \leq k \leq n$), where k denotes the k^{th}-operation to be processed on M_i.

The processing unit of a job on a machine is denoted as operation O_{jh}. Every operation O has at most two direct predecessor operations, a job predecessor PJ_o and a machine predecessor PM_o. Note that the first operation of a machine sequence has no PM_o whereas the first operation of a job has no PJ_o. Analogous every operation has at most two direct successor operations, a job successor SJ_o and a machine successor SM_o. The last operation of a machine sequence has no SM_o and the last operation of a

job has no SJ_o. An operation is called schedulable if both PJ_o and PM_o are already scheduled. The objective is to find a processing order ν such that the total makespan is minimized. An intuitive way of representing a JSP schedule is the Gantt-Chart.

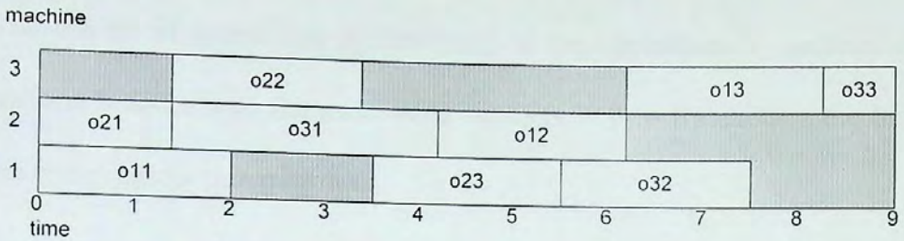

Figure 3: Example of a Gantt-Chart representation of a 3x3 job shop problem.

A schedule is called active, if the makespan improvement cannot be gained even by changing any of the processing orders. A non-delay schedule is given if no machine is kept idle when it could start processing some operations. We can state that the class of passive schedules includes active ones. Furthermore the class of active schedules includes non-delay schedules. Concerning a minimal makespan, at least one optimal schedule is an active schedule but not necessarily a non-delay schedule.

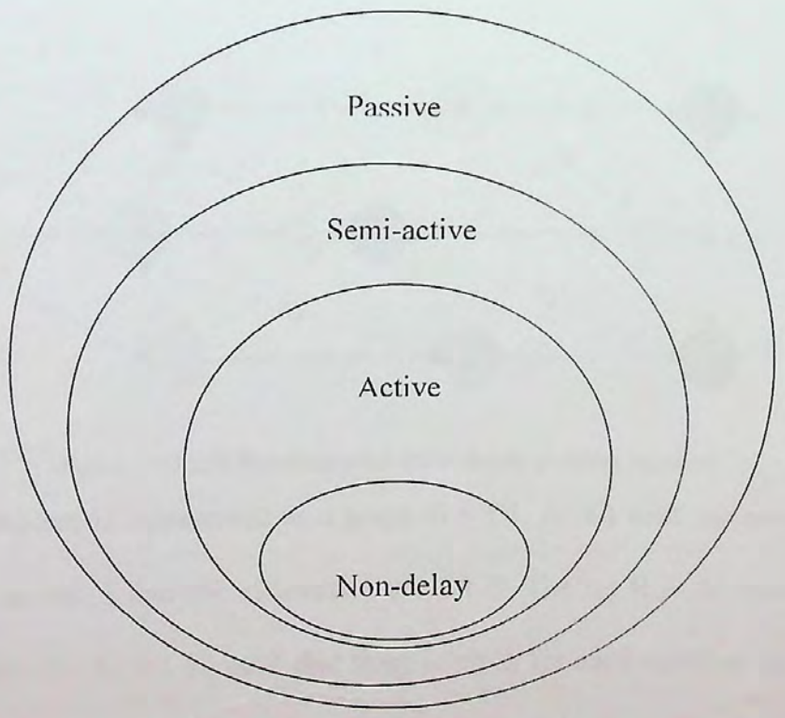

Figure 4: Classification of the various schedules.

If the starting time of an operation cannot be delayed without causing a deterioration of the makespan, it is called a critical operation.

Roy and Sussman first introduced the graph representation from 1964.

Let V be the set of operations. Additionally to the operations V contains two dummy operations b and c with the processing times $p_b = p_c = 0$ denoting the begin and end of the entire production programs.

In order to express the precedence of operations regarding jobs and machines, the sets A and E are introduced.

- Set A denotes the technological constraints as pairs of successive operations v, w $\in$ V, such that $v = PJw \wedge w = SJv$.
- The set E consists of m subsets E_i denoting pairs of operations to be processed on M_i, such that $v = PMw \wedge w = SMv$.

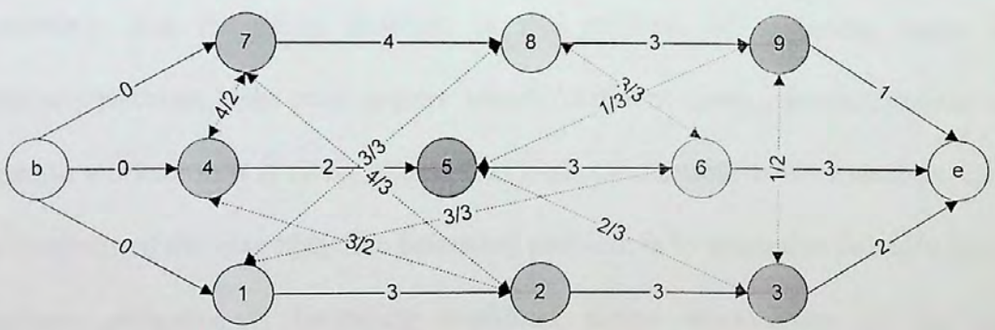

Figure 5: Graph Representation for a simple problem instance.

The problem is represented as a graph G = (V, AUE) with the node set V, the conjunctive arc set A and the disjunctive arc set E. The set E is decomposed into a subsets E_i with $E = U_i = 1\ E_i$, such that there is on E_i for each machine M_i. The terms 'node' and 'operation' and the terms 'arc' and 'constraint' are used synonymously.

The arcs in A and E are weighted with the processing time of the operation representing the source node v of the arc (v, w). The different gray shadings denote the various machines on which the operations are to be processed.

Assembly Line Balancing

Assembly line balancing problems consist of distributing work required to assemble a product in mass or series production on an assembly line among a set of workstations. Several constraints and different objectives may be considered.

An assembly line is a sequence of workstations, connected together by a material handling system, which is used to assemble components into a final product. The assembly process consists of a sequence of tasks or work-elements. A task consists of some elemental operations, which are tied together because of the use of a common tool or fixture. Accordingly, tasks cannot be sub-divided and must be completed at their assigned workstation. The tasks in an assembly process are typically ordered i.e. there may be precedence requirements that must be enforced. The assembly line balancing problem is the problem of assigning tasks to workstations. Because tasks may require widely different times, the assignment of task-times to workstations is rarely equal. This leads to idle time at workstations. One of the objectives of the assembly line balancing problem is to minimize this idle time. A secondary objective is balancing workload across workstation so that no workstation has excessively high or low workload.

The assignment of tasks to workstations is done to ensure that the assembly line can meet the demand rate. Thus each workstation is given a fixed amount of time to complete its task. Depending on the demand rate and task times it may sometimes be necessary to duplicate one or more workstations. Figure 6 gives a schematic of a

simple serial system and one with a parallel workstation. When the demand is high enough it is not uncommon to duplicate the entire assembly line. This has the advantage of shortening the assembly line but may require more equipment and tooling. Parallel assembly lines have other advantages as well. Because each workstation has a larger amount of time to complete its tasks, more tasks can be assigned to the station thereby enriching the work content. Also if equipment problems occur at a station other lines can continue to run. A single serial line would have to be shutdown whenever there is failure at any workstation.

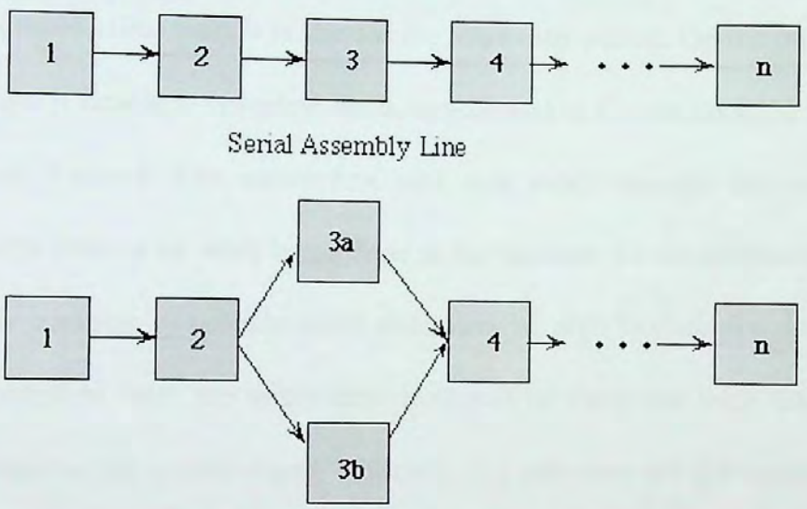

Figure 6: Shows a schematic of a simple serial system and one with a parallel workstation.

While the assembly line balancing problem is typically solved after the assembly process has been designed, it is a good practice to revisit the assembly process to redesign tasks to remove any problem areas. Thus, a bottleneck station could be provided more slack by redesigning some of the tasks and their associated tooling.

Assembly Line Types

Since the task times allotted to workstations may be unequal, parts are produced at different speeds on the line. This may also happen due to the variability of task times. Accordingly, stations may either be starved or a queue may build up in front of a station. To regulate the flow of parts assembly lines are often paced. In a paced line each workstation is a given a fixed amount of time called the cycle time C. The material handling system is designed so that after every C time units the material handling system indexes advancing the part to the next station. If a workstation finishes in less than C time units it is idle for the remaining period. On the other hand, if the workstation is unable to complete the assigned tasks in C time units the part will still be indexed forward. The incomplete part will index through the remaining workstations with little or no work being done at the stations. To ensure that this does not happen it is common to provide some slack time at each workstation so that the chances of incomplete work are minimized. It should be clear that such slack times are more important at the earlier stages. Typically, if a part does not get completed on the assembly line it is taken to a repair and rework area off line for completion. It is not unusual for a stack of such repaired parts to be kept handy to feed the line whenever an empty slot develops on the line.

If there is too much variability in the task times, it is preferable to have unpaced or asynchronous lines. In such lines each station works at its own pace and advances the part to the next station whenever it completes the assigned tasks.

The demand for a single product is often not sufficient to justify an independent assembly line. In such cases it is common to design mixed model assembly lines. Typically, the products being made on the assembly line tend to have very similar

tasks and precedence diagrams. Thus, we may design a single line for the assembly of a product that has several different options on it. In designing such lines we have to be careful to design such that there is an overall optimal solution and then operate the line so that we appropriately sequence the different models on the line to ensure a smooth workflow.

Assembly Line Balancing Problem Types

The simple assembly line balancing problem (SALBP) is relevant for straight single product assembly lines where only precedence constraints between tasks are to be considered.

Type 1 of this basic problem (SALBP-1) consists of assigning tasks to workstations such that the number of stations is minimized for a given production rate. Type 2 (SALBP-2) is to maximize the production rate, or equivalently, to minimize the sum of idle times for a given number of stations. A more general type (SALBP-G) is obtained by minimizing the sum of idle times subject to varying production rates and numbers of stations.

The U-line balancing problem (UALBP) considers the case of U-shaped (single product) assembly lines, where stations are arranged within a narrow U. As a consequence, workers are allowed to work on either side of the U, i.e. on early and late tasks in the production process simultaneously. Therefore, modified precedence constraints have to be observed. By analogy with SALBP, different problem types can be distinguished.

Mixed model assembly lines produce several models of a basic product in an intermixed sequence. Besides the mixed model assembly balancing problem (MALBP), which has to assign tasks to stations considering the different task times

for the different models, the mixed model sequencing problem (MSP) is relevant. MSP has to find a sequence of all model units to be produced such that inefficiencies (work overload, line stoppage, off-line repair etc.) are minimized.

Comprehensive descriptions of all problem types outlined above can be found in the [Scholl 1999].

Bin Packing Problem

Packing of items into boxes or bins is a recurring task in distribution and production. There is a large variety of different packing problems. Each problem differs from the other regarding the size and shape of items as well as the form and capacity of bins. Similar problems concern cutting of pieces into particular smaller ones so as to minimize the wastage of material and scheduling of identical parallel processors so as to minimize the total completion time. One of the basic packing problems is known as the one-dimensional bin packing problem. It is to pack a given set of items having different sizes into a minimum number of equal-sized bins.

Practical applications of the bin packing problem include stock cutting (a two-dimensional version of the problem), processor scheduling, commercial scheduling for television and music scheduling for radio stations, and the obvious.

The bin packing problem is an NP-hard problem. Therefore, there is not a known polynomial-time algorithm that produces optimal packings in every case. However, there are many known algorithms that approximate optimal packings. They are divided into two major classifications: on-line and off-line.

An on-line algorithm for the bin packing problem is an algorithm in which the items arrive in some order and must be assigned to a bin as soon as they arrive,

without knowledge of the remaining items. There cannot be any intermediate space for the items (or packages) to be placed; they must be placed into a bin immediately.

Four common simple on-line algorithms are First Fit (FF), Best Fit (BF), Next Fit (NF), and Worst Fit (WF). [Coffman et al. 1997]

- FF takes an item and places it into the first bin that will accommodate it.
- BF takes an item and places it into the bin that provides the best fit for it.
- NF only keeps one bin "open" (usable) at a time. It takes an item and checks the bin to see if it will fit. If it will, it packs it in; if not, it "closes" the bin, "opens" a new one, and packs the item into the newly opened bin.
- WF takes an item and places it into the bin with the most room.

In each of these algorithms, if a bin does not exist that the current package can fit into, a new bin is created and the package is placed into it.

An off-line algorithm is basically any algorithm that does not qualify as an on-line algorithm. These algorithms allow intermediate space to be used to hold the packages before they are packed.

There are many off-line algorithms. Four of the most common stem from the four on-line algorithms previously mentioned. The four algorithms are known as First Fit Decreasing (FFD), Best Fit Decreasing (BFD), Next Fit Decreasing (NFD), and Worst Fit Decreasing. [Coffman et al. 1997] These four algorithms sort all the packages in non-increasing order then pack them like their non-sorting counterparts. By doing this, they look to improve the worst-case performance ratio. Sorting the list generally does improve the worst-case ratio; however, it does not in NFD's case.

There are also variations of these packing schemes that look at specific package sizes to pack, or that look at pairs of items when deciding what to pack in a bin. Examples of these are Modified First Fit Decreasing and Best Two Fit.

In addition, there are some approximation schemes that perform better than these basic ones. However, their polynomial runtimes are much higher than these, and therefore are not very useful in the average case.

Related Work

The difficulty of the scheduling problems has lead to a long history of techniques emanating from the fields of AI and OR that provide approximate solutions to fairly general classes of problems or exact solutions to highly specific and restricted problems. The former are the more common and tend to rely on the use of heuristics, some form of stochastic optimization technique, or a mixture of both.

Variations of the resource-constrained scheduling problem have been proposed, implemented, and evaluated for over fifty years. The solution methods form two distinct classes: exact methods and heuristic methods. These classes may be categorized further into stochastic and deterministic approaches. Exact methods are guaranteed to find a solution if it exists, and typically provide some indication if no solution can be found. Heuristic solutions may have no such guarantee, but typically assure analytically some degree of optimality in their solutions. Stochastic methods include probabilistic operations so that they may never operate the same way twice on a given problem (but two different runs may result in the same solution). Deterministic methods operate the same way each time for a given problem. Many hybrid methods exist that combine the characteristics of these classes.

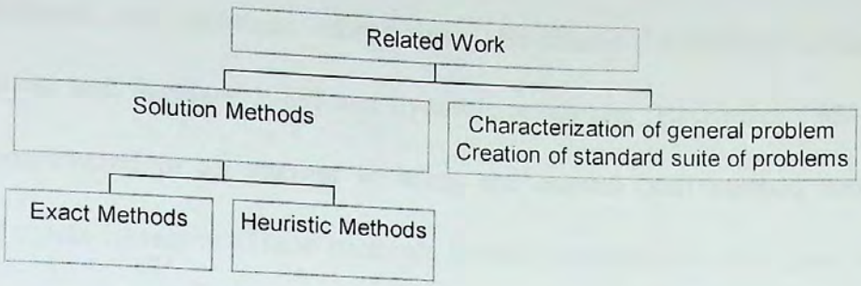

Figure 7: Classification of solution methods.

In addition to solution methods, recent work has focused on characterization of the general problem and creation of standard suites of problems on which to test algorithms. Even specialized instances of the resource-constrained scheduling problem are very complicated, so simply formulating the problem is non-trivial. As a result, most of the published problems are specialized, simplified, instances of the general problem. [Wall 1996]

Exact Solution Methods

Exact methods are guaranteed to find the optimal solution, but typically become impractical when faced with problems of any significant size or large sets of constraints.

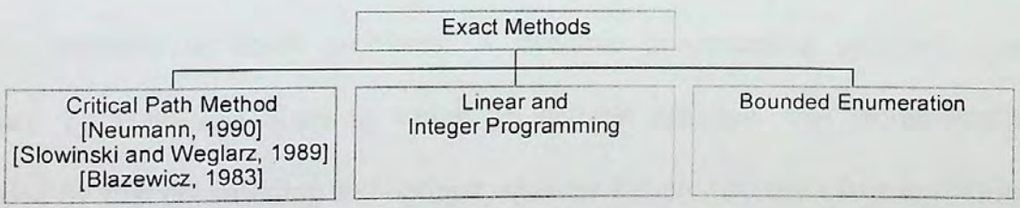

Figure 8: Classifications of the exact solution methods

The critical path method (CPM) provides the resource-unconstrained schedule for a set of precedence-constrained activities with deterministic durations. It gives the shortest possible makespan assuming infinite resources. Although useful for obtaining a rough idea of the difficulty of executing a plan, the critical path method does not

consider temporal or resource constraints. Stochastic variations [Neumann 1990][Slowinski and Weglarz 1989] and dynamic variations [Blazewicz 1983] have also been constructed in an attempt to bring the critical path method modeling assumptions closer to reality. These methods include probabilistic estimates of task duration.

Many scheduling problems can be formulated in traditional linear or integer programming form, but only if significant simplifications are made. Patterson presented an overview of optimal solution methods for project scheduling [Patterson 1984], and Demeulemeester and Herroelen published a more recent survey [Demeulemeester and Herroelen 1992].

In general, exact methods depend on characteristics of the objective function (e.g. strictly integer values) and specific constraint formulations (e.g. only single-mode tasks). As Lawrence Davis noted, many of the constraints commonly found in real scheduling problems do not lend themselves well to traditional operations research or math programming techniques [Davis 1985]. In addition, the linear programming formulations typically do not scale well, so they can be used only for specific instances or small problems. A dynamic programming approach was described by Held and Karp in which an optimal schedule was incrementally developed by first constructing and optimal schedule for any two tasks then extending that schedule by adding tasks until all tasks have been scheduled [Held and Karp 1962].

Many solution methods search a decision tree generated from the precedence relations in the project plan. The root of the tree corresponds to the first task. The second level of the tree is the set of tasks that can be scheduled once the first task has

been scheduled, and so on. The final tree thus represents a precedence-feasible set of task sequences. Any one of the root-to-leaf sequences can then be passed to a schedule generator. Alternatively, the sequence of tasks can be scheduled directly if the tree generation/pruning algorithm also considers resource constraints. The search consists of traversing the tree until the best root-to-leaf path is found. Enumerative methods are typically bounded using heuristics in order to reduce the size of the tree.

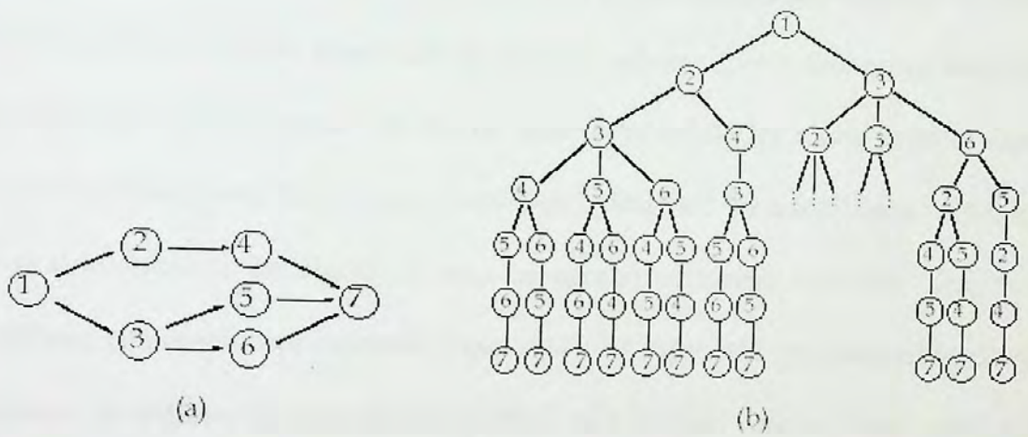

Figure 9: Generation of a tree of precedence-feasible sequences from a project plan (or work order). The precedence constraints are shown in (a) for a project (or work order) with 7 tasks. The tree of precedence-feasible sequences for scheduling the 7 tasks is shown in (b).

Variations of branch and bound solution methods were first proposed in the 1960s [Lawler and Wood 1966][Johnson 1967][Muller-Merbach 1967]. Stinson's branch and bound approach generated a tree by scheduling activities starting with the first task then adding a node to the tree for each task that could be scheduled based upon precedence and resource constraints [Stinson et al 1978]. Bounds based on partial schedules were used to prune the search tree. The heuristic for expanding the tree used a vector of six measures.

More recently, Sprecher, Kolisch, Drexl, Patterson, Demeulemeester, and Herroelen have refined the pruning algorithms so that optimal solutions to single-

mode project scheduling problems of about 100 tasks and multi-mode project scheduling problems of about 10 tasks can be found in less than a few seconds on personal computers. [Kolisch 1995][Sprecher and Drexl 1996]

Stefan Bock and Otto Rosenberg proposed a parallelized version of the Branch and Bound procedure for SALBP-1. Although the searching-tree of the sequential algorithm is extremely unbalanced the new distributed algorithm reaches optimal speedups if the problem-size is large enough. Besides even over linear speedups for some instances the algorithm implemented on a 1024-transputer-network is nearly 800 times faster than the sequential version. To achieve this a new work balancing technique has been designed that can be used on principle for many parallel depth-first-search Branch and Bound algorithms. Being designed for a local area-network of personal computers, the algorithm uses no special transputer attribute. Due to its predefined communication structure it can easily be supported by practical hardware-solutions. In addition the algorithm supports a fault tolerant concept. So if some node fails the algorithm can go on working without loosing any solution, which is very useful in a network of more than one hundred personal computers.

Scholl and Klein designed a bi-directional branch and bound procedures, which clearly outperforms the currently best known algorithms EUREKA, OptPack and FABLE. [Scholl and Klein 1996] It contains a clever branching strategy and various known and new lower bounding procedures.

Scholl and Klein proposed an exact solution procedure, called Progress, for the generalized resource-constrained project-scheduling problem (GRCPSP). This NP-hard problem extends the well-known resource-constrained project scheduling problem (RCPSP) by introducing minimum time lags between project activities and

by considering dynamic resource availability during the planning horizon. Though these extensions are of high practical importance, only a few exact solution procedures have been presented in the literature so far, among which the one of Demeulemeester and Herroelen is the most efficient. By improving their dominance rules and combining them with a recently introduced branching strategy, the so-called Local Lower Bound Method the new exact solution procedure Progress is developed, which clearly outperforms all the other procedures. Including new dominance rules and lower bounding techniques further increases its efficiency. [Klein and Scholl 1998]

As noted by Sprecher and Drexl, enumerative methods cannot solve large problems; the tree is simply too big Although significant progress has been made in the pruning techniques, branch and bound methods are still limited to less than one hundred activities or even fewer in the multi-modal cases, and they still require special heuristics to accommodate variations in resource constraint formulations.

Heuristic Solution Methods

Whereas exact solution methods are guaranteed to find the optimal solution (if one exists), heuristic methods sometimes find optimal solutions, but more often find simply "good" solutions. Heuristic methods typically require far less time and/or space than exact methods. The heuristics specify how to make a decision given a particular situation; heuristics are rules for deciding which action to take.

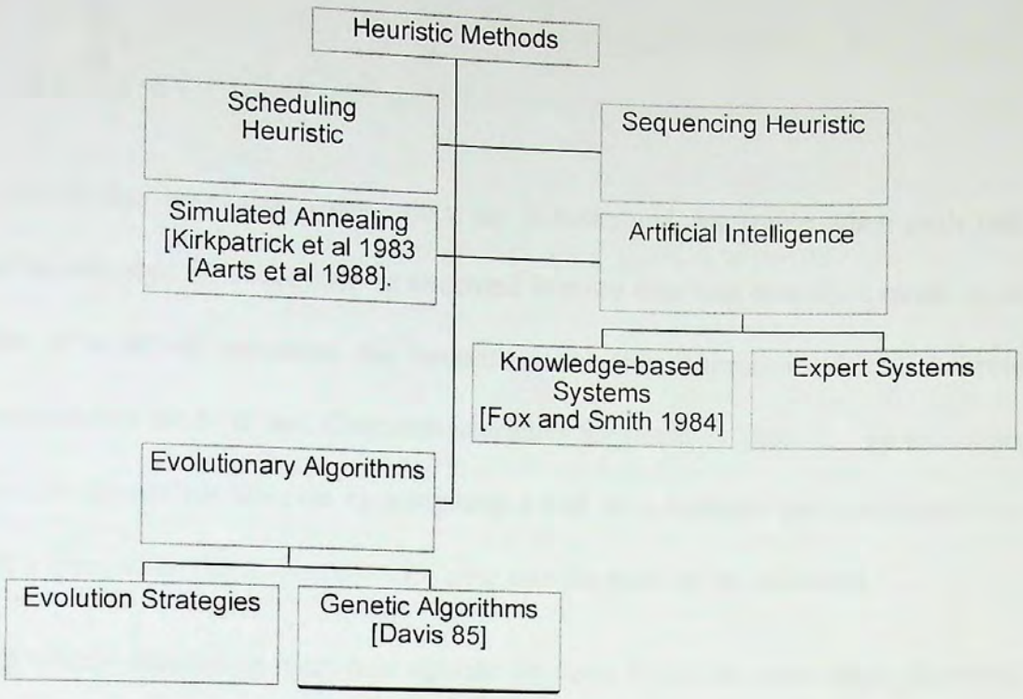

Figure 10: Classification of the heuristic solution methods.

Heuristics in scheduling are often referred to as scheduling rules or dispatch rules. The definition of these rules is often quite complex, and most are tailored for a specific type of problem with a very specific set of constraints and assumptions. Heuristics may be deterministic - they end up with the same result every time - or they may be stochastic – each time they are run they may produce a different result. They may execute one rule at a time, or they may be capable of parallel decisions. Hybrid algorithms may combine multiple heuristics.

The quality of the solution obtained from a heuristic can be determined by comparing the solution with a lower bound. For the assembly line balancing problem a simple lower bound on the number of stations can be calculated by dividing the total processing time by the cycle time.

$$LB = \frac{\sum_{j=1}^{N} t_j}{C}$$

Scheduling heuristics operate on a set of tasks and determine when each task should be executed. If a task may be executed in more than one execution mode or on any one of a set of resources, the heuristic must also determine which resources and/or execution mode to use. Common heuristics are listed in Table 1. The scheduler enforces constraint satisfaction by assigning a task to a resource (or a resource to a task) at a time when the resource is available and the task can be executed.

Whereas scheduling heuristics operate on tasks to decide when they should be executed, sequencing heuristics determine the order in which the tasks will be scheduled. These heuristics are often used in combination with decision trees to determine which part of the tree to search or to avoid. For example, limited discrepancy search with backtracking has been used by William Harvey and Matthew Ginsberg to very effectively solve some classes of scheduling problems when the sequence for scheduling tasks is structured as a decision tree [Harvey and Ginsberg 1994].

heuristic	what it does
MIN SLK	choose the task with the smallest total slack
MIN LFT	choose the task with the nearest latest finish time
SFM	choose the execution mode with the shortest feasible duration
LRP	choose the execution mode with the least resource proportion

Table 1: Some commonly used scheduling heuristics. Dispatch rules (a form of scheduling heuristic) decide which resources should receive tasks as they come in to a shop. [Wall 1996]

The idea behind the Ranked Positional Weight heuristic is to first assign the tasks, which have long chains of succeeding tasks. The length of the chain can be measured either by the number of successor tasks or by the sum of the task times of the successor tasks. In the RPW method, we define the sum of task times of the successor tasks as the Positional Weight of the task. The tasks are ranked in descending order of the Positional Weights with ties broken arbitrarily. The tasks are then picked in their ranked order and assigned to workstations if all predecessors of the task have already been assigned and the task fits in the remaining time on the workstation. If a task does not fit in the remaining time on a workstation it is skipped and the next task in the ranked order is selected. If no task fits in the workstation, the workstation is closed and a new workstation is opened. The tasks are then scanned from the beginning of the list and an attempt is made to assign unassigned tasks to the new workstation. The process is repeated until all tasks are assigned.

While the RPW heuristic gives reasonably good solutions, it only gives a single solution. Often criteria other than minimum number of stations may have to be considered. It is therefore important to obtain several solutions. One way to achieve this is to use a randomized algorithm. In such an algorithm, several possible solutions can be generated by randomly selecting a feasible task from the available tasks instead of selecting the "best" task according to some criterion. The Computerized Method for Sequencing Operations on Assembly Lines (COMSOAL) method uses this strategy. At each stage of the algorithm, the set of tasks that can be feasibly assigned and which fit in the remaining time on the current workstation is determined. The task to be assigned is then randomly selected with each task having the same probability of being selected

Artificial intelligence approaches to scheduling can be grouped as either expert systems or knowledge-based. Both are structured heuristic methods that differ in the way they control the application of their application-specific heuristics.

Expert systems consist of a rule base, a snapshot of the current solution, and an inference engine. The inference engine determines how the if-then rules in the rule base are applied to the current solution in order to execute the search. The rule base may be expanded as the solution progresses. The rule base is tailored explicitly to a specific problem, so expert systems are typically highly specialized.

Knowledge-based systems typically split the problem into sub-problems or different views. "Agents" are defined, each of which is concerned with a particular aspect of the solution. Each agent nudges the solution in the direction of most concern to that agent. Variations to the algorithms include combinations of micro and macro modifications to solutions as well as the types of attributes to which the agents are configured to respond. Hildum distinguished between three commonly known artificial intelligence solutions, ISIS (Intelligent Scheduling and Information System) [Fox and Smith 1984], OPIS (Opportunistic Intelligent Scheduler) [Smith and Ow 1985], and MicroBOSS (Micro-Bottleneck Scheduling System) [Sadeh 1991], based upon the rules they used to guide their searches. Hildum noted that his own method, DSS (Dynamic Scheduling System), is basically a multiple attribute, dynamic heuristic approach that focuses on the most urgent unsolved problem at any given time.

Simulated annealing approaches require a schedule representation as well as a neighborhood operator for moving from the current solution to a candidate solution.

Annealing methods allow jumps to worse solutions and thus often avoid local sub-optimal solutions [Kirkpatrick et al 1983].

Aarts, Laarhoven, and Lenstra described one of the first simulated annealing approaches to scheduling problems [Aarts et al 1988].

Chikong Huang presents in his paper a searching model combining a traditional line balancing heuristic and the simulated annealing procedure. The objective function of this searching model is based on the system utilization. A hypothetical example is also demonstrated by using the proposed searching model to find a near optimal production line. [Huang 1997] The numerical result indicated that the proposed searching model could effectively improve the system utilization.

Lawrence Davis made one of the earliest suggested uses of genetic algorithms for scheduling. In his paper [Davis 1985], Davis noted the attractiveness of using a stochastic search method due to the size of the search space and suggested an indirect representation in which the genetic algorithm operated on a list which was then decoded to form the actual schedule. In particular he noted the importance of maintaining feasibility in the representation.

representation	characteristics	reference
order 6 order 1 order 4 ...	list of orders to be scheduled in a job-shop	[Syswerda 91]
1 0 1 1 0 0 1 1 1 1 0 0 ...	binary representation in which each bit determines which order of a pair should be executed first on a given machine	[Nakano 91] [Cleveland 89]
order 2 plan X order 1 plan Q order 4 plan B ...	list of (order, plan) pairs	[Bagchi 91]
order 5 plan A op1 : m3 op2 : m5 op3 : m3 order 2 plan X op1 : m1 op2 : m7 order 1 plan A op1 : m4 op2 : m6 op3 : m7 ...	list of (order, plan, resource) tuples	[Bagchi 91]
station 1 0 : o1 wait o4 idle 60 o2 wait o1 wait o4 ... station 2 0 : o4 wait o4 idle 60 o5 wait o6 idle	list of order/time preferences for each workstation	[Davis 85]
order 5 machine 3 00:00 order 2 machine 6 01:05 order 1 machine 2 00:15 ...	list of order/machine/time tuples	[Kanet 91]
order 7 Op7B1 : m9 : [10,15] Op7B2 : m3 : [16,17] Op7B3 : m6 : [18,22] order 9 Op9A1 : m9 : [10,15] Op9A2 : m3 : [16,17] ...	list of complete order information	[Bruns 93]
task 2, mode 5 order 5 01:00 - 04:30 task 5, mode 1 order 1 00:30 - 00:50 ...	list of (activity-mode, order, start-finish-time) tuples	[Mori and Tseng 96]
project 1 activity 2 mode 2 project 1 activity 3 mode 4 project 2 activity 1 mode 1 ...	list of (project, activity, mode) tuples	[Tseng and Mori 96]

Table 2: Summary of various genetic algorithm formulations. The references in the table are representative of the type of solution; this table does not contain an exhaustive list of published works. Most of the representations are order-based, i.e. the order in which the items appear in the list is a part of the problem structure. [Wall 1996]

CHAPTER III GENETIC ALGORITHMS

Overview

Genetic algorithms are a stochastic heuristic search method whose mechanisms are based upon simplifications of evolutionary processes observed in nature. Since they operate on more than one solution at once, genetic algorithms are typically good at both the exploration and exploitation of the search space. Goldberg [Goldberg 1989] provided a comprehensive description of the basic principles at work in genetic algorithms, and Michalewicz [Michalewicz 1994] described many of the implementation details for using genetic algorithms with various data types.

Most genetic algorithms operate on a population of solutions rather than a single solution. The genetic search begins by initializing a population of individuals. Individual solutions, or genomes, are selected from the population, then mate to form new solutions. The mating process, typically implemented by combining, or crossing over, genetic material from two parents to form the genetic material for one or two new solutions, confers the data from one generation of solutions to the next. Random mutation is applied periodically to promote diversity. If the new solutions are better than those in the population, the individuals in the population are replaced by the new solutions.

Genetic algorithms operate independently of the problems to which they are applied. The genetic operators are heuristics, but rather than operating in the space defined by the problem itself (the solution-space, or phenotype-space), genetic operators typically operate in the space defined by the actual representation of a solution (representation-space, or gene-space). In addition, genetic algorithms include other heuristics for determining which individuals will mate (selection), which will

survive to the next generation (replacement), and how the evolution should progress (the overall algorithm).

The genetic algorithm includes some representation-specific operators and some representation-neutral operators. The initialization, mating (typically implemented as crossover), and mutation operators are specific to the representation. Selection, replacement, and termination are all independent of the representation. In the context of the scheduling problem, the representation is specific to scheduling, but a general representation may be used with many different variations of the scheduling problem.

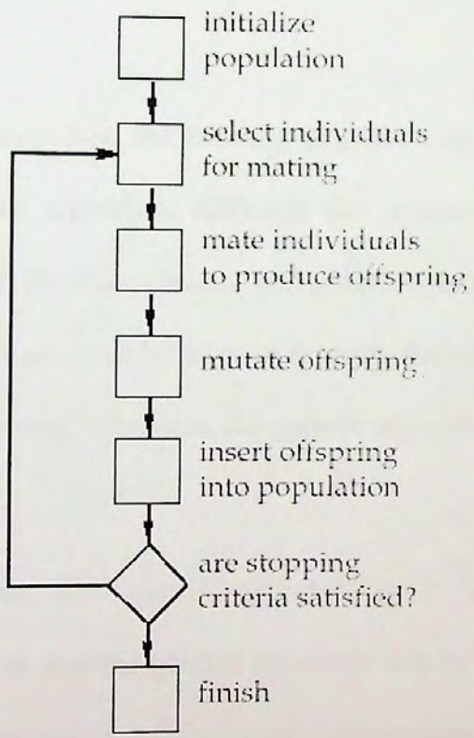

Figure 11: Generic genetic algorithm flowchart. Many variations are possible, from various selection algorithms to a wide variety of representation-specific mating methods. Note that there is no obvious criterion for terminating the algorithm. Number-of-generations or goodness-of-solution are typically used.

In traditional schedule optimization methods, the search algorithm is tightly coupled to the schedule generator. These methods operate in the problem space; they

require information about the schedule in order to search for better schedules. Genetic algorithms operate in the representation space. They care only about the structure of a solution, not about what that structure represents. The performance of each solution is the only information the genetic algorithm needs to guide its search. For example, a typical heuristic scheduler requires information about the resources and constraints in order to decide which task should be scheduled next in order to build the schedule. The genetic algorithm, on the other hand, only needs to know how "good" a schedule is and how to combine two schedules to form another schedule.

Having said that, many hybrid genetic algorithms exist which combine hill-climbing, repair, and other techniques which link the search to a specific problem space.

Proper choice of representation and tailoring of genetic operators is critical to the performance of a genetic algorithm. Although the genetic algorithm actually controls selection and mating, the representation and genetic operators determine how these actions will take place. Care must be taken to properly define the representation, operators, and objective function, otherwise the genetic algorithm will perform no better than a random search.

Gruninger showed that the genetic operators must effectively balance exploration and exploitation so that the genetic algorithm will be able to both avoid local minima/maxima in global search and find small improvements in local search.

Some genetic algorithms introduce another operator to measure similarity between solutions in order to maintain clusters of similar solutions. By maintaining diversity in the population, the algorithms have a better chance of exploring the search space and avoid a common problem of genetic algorithms, premature convergence.

After a population has evolved, all of the individuals typically end up with the same genetic composition; the individuals have converged to the same structure. If the optimum has not been found, then the convergence is, by definition, premature. In most cases, further improvement is unlikely once the population has converged.

The similarity measure is often referred to as a distance function, and these genetic algorithms are referred to as speciating or niching genetic algorithms. The similarity measure may be based upon the data in the genome (genotype-based similarity), it may be based upon the genome after it has been transformed into the problem space (phenotype-based similarity), or it may integrate some combination of these.

Genetic Algorithms Approaches to Scheduling

Early Work

One of the earliest published works on the application of genetic algorithms to scheduling is that by Davis [Davis 1985]. While his paper merely outlines a basic scheme applied to a highly simplified toy problem in flow-shop scheduling (all jobs involve the same processing operations applied in the same order), it contains some interesting and worthwhile material. Davis conjectured that a genetic algorithm might be designed that could handle the constraints but by virtue of its stochastic nature would avoid poor local minima.

Davis's solution was to use a less literal genotype that was amenable to crossover but required a decoding phase to turn it into a legal solution to the problem. The genotype was a list of preference lists, one for each machine. A preference list consisted of an integer that specified the time the machine should start processing,

followed by a permutation of the jobs available and the elements "wait" and "idle". Whenever it had to be decided which operation a machine should perform next, the first available job from its preference list was chosen. The element "idle" forced the machine to remain idle, giving preference to other machines. The element "wait" prevented the machine from processing further along its preference list. The genetic operators employed were as follows: a crossover that exchanged preference lists for selected machines; a scramble operation that randomly reordered members of a preference list; and run-idle, a heuristic operator that inserted "idle" as the second element of the preference list of a machine that had to wait more than an hour for jobs to become available. Each operator was applied probabilistically. The evaluation function used summed the costs of running the flow-shop for five hours with the schedule represented by an individual. Penalty costs were added if jobs were not completed during this interval. [Husbands 1994].

During the early 1980s Fourman experimented with genetic algorithm methods for tackling layout problems in VLSI, problems very closely related to scheduling. He addresses a symbolic layout problem in which rectangular blocks of fixed sizes, connected by fixed width lines, and subject to various topological and geometric constraints, had to be arranged so as to minimize the total area while violating as few design rules as possible. His genotypes were lists of symbols representing geometric and topological constraints on the positioning of the blocks and connecting lines that were used to determine a layout by feeding them through a deterministic procedure. Standard crossover, inversion and mutation were readily adapted to operate on these lists. The results obtained were promising but not outstanding. Fourman made some interesting suggestions about using a refined version of the genetic algorithm systems in which the designer could intervene in the selection process by providing `hints'

based on hard to formalize design knowledge. This sort of approach might lead to the semi-automation of complex design tasks not amenable to other optimization techniques, because of these unformalizable aspects.

At about the same time Smith and Davis devised a hybrid algorithm, based on a genetic algorithm, for bin packing [Smith 1985], that is packing a set of regularly shaped boxes into a fixed space according to some packing density criteria. The work was similar in spirit to Davis's scheduling research described earlier. Again the problem is closely related to scheduling, it is essentially identical to sequencing jobs on a single machine. The genotypes used were simple integer lists determining the order in which the boxes were presented to a deterministic algorithm that went on to do the packing. By combining the GA with the bin-packing algorithm in this way, rather than attempting to explicitly represent the whole layout on the genotype, they produced high quality solutions two orders of magnitude faster than with dynamic programming methods.

During the mid-eighties Husbands, Mill and Warrington successfully applied a genetic algorithm to the problem of process plan optimization [Husbands 1987]. This is a problem closely related to scheduling in which an optimal set of operations and operation orderings must be found for a single job. Good solutions can sometimes be found using heuristics in conjunction with branch and bound search, but the heuristics are not robust: circumstances can easily conspire to fool them into poor local minima. A genetic algorithm was developed that performed robustly over a very wide range of situations and consistently found better solutions than branch and bound and other OR techniques.

General Sequencing Applications

Before going on to describe later developments, it is worthwhile summarizing results in the more general application of genetic algorithms to sequencing problems, mainly the TSP, a pure sequencing problem which has a close affinity with scheduling problems, particularly flow-shop scheduling. The task is to find the shortest route through a set of cities, visiting each once only and returning to the starting point. An obvious genotype is a permutation of a list of integers representing the cities. Using this representation, simple crossover would produce illegal tours most of the time, with some cities from the parents represented twice and some not at all. Early efforts by Goldberg [Goldberg 1995] and Grefenstette [Grefenstette 1985] overcame this problem by correcting the offspring tours so that the duplicate cities were replaced by the omitted cities or otherwise eliminated.

Whitley et al. produced better results by developing a representation and recombination operator that manipulated edges (links between cities) rather than the cities themselves [Whitley 1990]. Whitley and his co-workers went on to use this work to develop a prototype production line scheduling system for a Hewlett-Packard board assembly facility.

Fox and McMahon published an insightful paper on genetic encoding and operators for sequencing problems [Fox 1991]. They introduced a bit string representation for sequences based on a Boolean matrix representation of ordering relationships. This scheme can be used for partial orderings as well as the total ordering required for the TSP.

Fox and McMahon compared these operators with Whitley's edge recombination and Goldberg's PMX operator among others. They were tested on a 30

city TSP. The operators performed very similarly in terms of quality of solution found. However, the union and intersection operators produced very good solutions in far fewer generations than the other operators although they were significantly more computationally expensive.

Gorges-Schleuter has produced good results for large TSP problems by incorporating local hill-climbing heuristics into a parallel genetic algorithm [Gorges-Schleuter 1989].

Later Developments in Scheduling with Sequential Genetic Algorithms

Cleveland and Smith investigated the use of genetic algorithms in scheduling a multi-stage flow line with non-standard characteristics. The problem they addressed is combinatorially complex so previous progress had only been made through the use of heuristics. Their intention was to examine genetic algorithms as an alternative approach. They studied three basic models: a pure sequencing version, which assumed that all jobs were available for release at the start of the scheduling horizon, and that work-in-progress cost are negligible; a model that included consideration of actual release times; and a model that also included work-in-progress costs.

The late eighties saw an explosion in the number of genetic algorithm researchers. Consequently there has been a fair volume of very recent work on scheduling. Some of this will be outlined now, before going on to more advanced techniques. Gabbert et al. successfully applied a genetic algorithm to transport (train) scheduling and routing. Unlike other approaches, they were able to use modifiable complex cost models, avoiding most of the standard simplifications. They are confident of being able to scale-up their prototype systems. This seems to be a good

example of exploiting the strengths of genetic algorithms to handle those aspects of a problem that make it unamenable to more traditional techniques.

Wren and Wren have done some very interesting preliminary work on applying genetic algorithms to the hard practical problem of bus driver scheduling [Wren 1990]. Using a straightforward genetic representation of the problem, but an involved and insightful recombination operator, they were able to find solutions as good as those produced by the best OR techniques.

Another significant result, also coming from the OR community, is that of Dorndorf and Pesch [Dorndorf 1992]. They use a genetic algorithm to find optimal sequences of local decisions rules to be used with OR search algorithms. For a range of static deterministic job-shop scheduling problems their hybrid algorithm was able to find shorter makespan quicker than Adams, Balas and Zawack's shifting bottleneck procedure and Laarhoven, Aarts and Lenstra's simulated annealing approach. These two techniques were generally regarded as the best available.

Mansour and Fox developed a hybrid genetic algorithm, making use of local hill-climbing and problem specific knowledge, for task allocation in multi-computers [Mansour 1991]. They found significantly better solutions than with a range of other techniques, although the genetic algorithm was computationally more expensive.

Nakano [Nakano 1991] tackled job shop scheduling with a genetic encoding similar to that employed by Fox and McMahon (described above) and closely related to Husbands' arbitrator strings described later. He used simple crossover with a fairly involved genetic repair mechanism to ensure legal offspring.

Syswerda describes a genetic algorithm based system for scheduling the use of laboratory equipment [Syswerda 1990]. He employed a genetic algorithm to find an

initial sequence of tasks to feed to a fairly sophisticated deterministic schedule builder such that near optimal schedules result. His genotype was simply a list representing a task permutation. He used various mutation operators: select two tasks at random, place the second before the first; select two tasks at random, interchange their positions; scramble a randomly chosen sub-list of the genotype. He experimented with order and position-based crossovers as well as Whitley's edge recombination operator. This hybrid approach, where the GA works in tandem with a deterministic search method, produced good results fast enough that it could take in the dynamic aspects of the problem and allowed rescheduling.

Reeves has done some preliminary experiments on applying genetic algorithms to stochastic flow shop problems [Reeves 1992]. Over a range of different problem instances his algorithm consistently outperformed two other techniques from the OR literature.

Ling was able to find good solutions to a large college time tabling problem by first using a heuristic-based algorithm to build a reasonable timetable, but with some constraints violated, and then applying a genetic algorithm to convert this into a solution with no constraints broken [Ling 1991].

Parallel Genetic Algorithms

From the very earliest days of its development the genetic algorithm's potential for parallelization, with all its attendant benefits of efficiency, has been noted. The availability of hardware has recently allowed significant progress in this direction. The standard sequential genetic algorithm uses global population statistics to control selection, so the processing bottleneck is evaluation. The earliest parallel models simply parallelized this phase of the sequential algorithm. Recently more

sophisticated parallel genetic algorithms have started to appear in which population can be viewed as being spread out geographically, usually over a 2D toroidal grid. All interactions, e.g. selection and mating, are local, being confined to small (possibly overlapping) neighborhoods. By doing away with global calculations, this allows the development of fine-grained highly parallel asynchronous algorithms. There is mounting evidence to suggest that such systems are more robust and faster (in terms of solutions evaluated) than other implementations. Highly parallel models can also result in powerful new ways of approaching optimization problems at the conceptual level.

Parasites and Sorting Networks

Danny Hillis, who lead the team that developed the Connection Machine [Hillis 1985], was the first to significantly extend the parallel genetic algorithm paradigm by showing how to develop a more powerful optimization system by making use of co-evolution. In his experiments Hillis takes full advantage of massive parallelism, routinely using populations of a million individuals evolving over tens of thousands of generations. Hillis uses a diploid genetic algorithm. Individuals are represented as pairs of number strings, analogous to the chromosome pairs of diploid organisms. The solutions, or phenotypes, coded for by the string pairs, are constructed in the usual way by interpreting fixed regions of the strings as coding for particular parameter values, or phenotypic traits. Discrepancies between the two strings of a pair are resolved according to some rule of dominance. He has also found that locally controlled selection is more robust than the simple global variety.

An interesting complex optimization problem that he has tackled using genetic algorithms, is the problem of finding minimal sorting networks for a given number of

elements. A sorting network represents a sorting algorithm in which comparisons and exchanges take place in some predetermined order. Finding good networks is of significant practical interest, bearing on the development of optimal sorting algorithms, switching circuits and network routing algorithms. The horizontal lines correspond to the elements to be sorted. The unsorted input is on the left and the sorted output is on the right. In between, comparison-exchanges of elements are indicated by arrows pointing from one element to another. A comparison-exchange of the i^{th} and j^{th} elements is indicated by an arrow from the i^{th} to the j^{th} line. Elements are exchanged if the element at the head of the arrow is strictly less than the element at the tail.

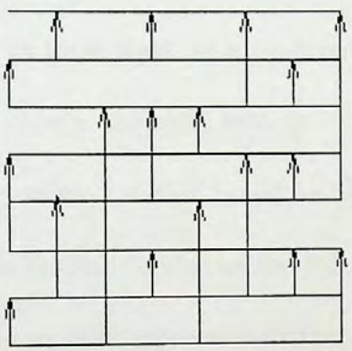

Figure 12: Sorting Network

The genotype of each individual consisted of a pair of bit string chromosomes. Each chromosome can be thought of as sixty eight-bit genes. Each gene consisted of two four-bit numbers representing elements to be compared and possibly exchanged. The phenotype (sorting network) is generated by traversing the chromosomes in a fixed order. If a pair of chromosomes have the same gene (comparison-exchange elements) at a particular site, then only that comparison-exchange pair is generated in the sorting network. If the genes are different, then both pairs are generated. In this way the network can contain between sixty and one hundred and twenty comparison-

exchanges. The phenotypes were scored according to how well they sorted. The measure used was the percentage of correct sorts performed on a sample of test cases. In the first set of experiments this sample was randomly generated. The best sorting network found contained sixty-five exchanges. This compares favorably with solutions found by other methods. In fact, for a number of years the best know solution used sixty-five exchanges.

Symbiosis and Emergent Scheduling

The underlying structure of many combinatorial optimization problems of practical interest is highly parallel. However, traditional approaches to these problems tend to use mathematical characterizations that obscure this. By contrast, the use of biologically inspired models casts fresh light on a problem and may lead to a more general characterization, which clearly indicates how to exploit parallelism and gain better solutions. This section describes a model based on simulated co-evolution that has been applied to a highly generalized version of the job shop scheduling problem, far more general than any of the models presented so far. Whereas Hillis's model is analogous to host-parasite ecology, this model is closer to a symbiotic ecology. That is, a number of separate species interacting in ways, which are to their mutual advantage.

In the standard model of manufacturing planning, process planning directly proceeds scheduling. A process plan is a detailed set of instructions on how to manufacture each part (process each job). This is when decisions are made about the appropriate machines for each operation and any constraints on the order in which operations can be performed. Very often completed process plans are presented as the raw data for the scheduler. Scheduling is essentially seen as the task of finding an

optimal way of interleaving a number of fixed, or maybe slightly flexible, plans which are to be executed concurrently and which must share resources. However, in many manufacturing environments there are a vast number of legal plans for each component. These vary in the orderings between operations, the machines used, the tools used on any given machine and the orientation of the work-piece on any given machine. They will also vary enormously in their costs. Instead of just generating a reasonable plan to send off to the scheduler, it is desirable to generate a near optimal one. Clearly this cannot be done in isolation from the scheduling. A number of separately optimal plans for different components might well interact to cause serious bottlenecks. Because of the complexity of the overall optimization problem, that is simultaneously optimizing the individual plans and the schedule, and for the reasons outlined in the introduction, up until now very little work has been done on it. However, recasting the problem to fit an `ecosystem' model of coevolving organisms has provided a promising new direction.

Husbands' model involves a number of different populations cooperatively solving a number of related problems allowing the emergent solution of a wider more complex problem. The genotype of each specie represents a feasible process plan for a particular component to be manufactured in the machine shop. Separate populations evolve under the pressure of selection to find near-optimal process plans for each of the components. However, their fitness functions take into account the use of shared resources in their common world (a model of the machine shop). This means that without the need for an explicit scheduling stage, a low cost schedule will emerge at the same time as the plans are being optimized.

The data provided by a plan space generator, whose operation is outlined later, is used to randomly generate populations of structures representing possible plans,

one population for each component to be manufactured. An important part of this model is the population of Arbitrators, again initially randomly generated. The Arbitrators' job is to resolve conflicts between members of the other populations; their fitness depends on how well they achieve this. It is important to note that the environment of each population includes the influence of all the other populations.

The genotype of a process plan organism can be represented as follows:

op1 m1 s1 op2 m2 s2 Gop3 m3 s3 op4 m4 s4 op5 m5 s5 G .....

Where op_i refers to the i^{th} operation in a plan, m_i to the machine to use for that operation and s_i to the setup. Operations with interdependencies are grouped together, each group being terminated by a special symbol (G in above example). As long as the group terminators are the only legal crossover points, the crossover operation will always produce legal plans. If crossover were to occur within a group, data for dependent operations would be split up and illegal plans would probably occur on recombination. The mutation operator is also fairly involved because the gene values are context sensitive due to the dependencies. This encoding encapsulates the network structures of the data produced by the plan space generator. Each op_i, m_i and s_i has associated with it finite sets of possible integer coded values. Because these sets are all quite different, bit string representations would be awkward and unnatural, hence so called real valued codes are used. The genotype is transformed into another form for interpretation by the fitness function, to be described later. This is to take into account the ordering aspect of the problem. There is a network of partial ordering constraints associated with each genotype (specie), the operations must be ordered in accordance with these.

The Arbitrators' genotype is a bit string which encodes a table indicating which population should have precedence at any particular stage (defined later) of the execution of a plan, should a conflict over a shared resource occur. There is one bit for each possible population pairing at each possible stage. Hence the Arbitrator genome is a bit string of length SN(N _ 1) = 2, where S = maximum number of stages in a possible plan and N = number of process plan organism populations. Each bit is uniquely identified with a particular population pairing and is interpreted according to the following function:

$$F(n_1, n_2, k) = g[KN(N-1)/2 + n_1(N-1) - n_1(n_1+1)/2 + n_2 - 1]$$

Where n_1 and n_2 are unique labels for particular populations, $n_1 < n_2$, k refers to the stage of the plan and g[i] refers to the value of the i^{th} gene on the Arbitrator genome. If f(n1; n2; k) = 1 then n1 dominates, else n2 dominates. By using pair wise filtering the Arbitrator can be used to resolve conflicts between any number of different species.

The cost functions for plan organisms involve two stages, for arbitrators just one. The first stage involves population specific criteria and the second stage takes into account interactions between populations. The first stage cost function for the process plan organisms, COST1 shown below, is applied to the genotype shown above after it has first been translated into a linearized format that can be interpreted sequentially.

$$COST1(plan) = \sum_{i=1}^{i=N} (M(m_i, i) + S(s_i, i, m_i))$$

Where si = setup used while processing i^{th} operation, mi = machine used for processing ith operation, S(si; i; mi) = setup cost for i^{th} operation, $M(m_i; i)$ = machining cost for i^{th} operation, N = number of operations to be processed and $M(m_i; i)$ has been previously calculated and is looked up in a table. Note that a setup cost is incurred every time a component is moved to a new machine or its orientation on the same machine changes. This function performs a basic simulation of the execution of the plan. Its input data is an ordered set of (machine, setup) pairs, one for each operation. The operations must be ordered in such a way that none of the constraints laid down by the planner are violated. Ordered sets of operations to be processed using a particular machine/setup combination are built up on a 2D grid. S(si; i; mi) governs the way in which the sets are built up on the grid. The operations in any set can be performed in isolation from those in any other set. Such a set will be referred to as a stage of a job in the rest of this paper. These sets themselves are ordered and the outcome is a process plan like the one shown below, where the integers in the square brackets refer to particular operations.

1) machine: 6 setup: 5 [0,3,5,7]

2) machine: 2 setup: 21 [1,8,12,19]

3) machine: 11 setup: 4 [2,4,6,9,13,15] ...etc

In fact COST1 provides a mapping from the process plan genotype to its phenotype: one of the plans illustrated above. Note that the setup cost is often considerably more (orders of magnitude) than the basic machining costs. The essential workings of COST1 is to sequentially process the transformed genome in order to group operations together in clusters which can then be scheduled as single units (stages). At the same time the final executable ordering of the operations is found, as

well as the basic machining costs. The mechanics of the function are fairly complex and will not be dealt with here, but see. The function involves a complicated interpretation of the genotype, more complex than at first sight seems necessary. However, it should be noted that a carefully chosen genetic encoding plus a complex, but computationally cheap, interpretation and costing function, allow very simple and cheap genetic operators to be employed which are capable of searching the combined ordering and machine selection space.

The second phase of the cost function involves simulating the simultaneous execution of plans derived from stage one. Additional cost is incurred for waiting and going over due dates. What happens when two plans want the same resource at the same time? Fixed precedence would be far too in flexible and random choices would be of no help. As already indicated, the most general and powerful solution developed was to introduce a new species, the Arbitrators, whose genetic code holds a table indicating which population had precedence at any stage. The Arbitrators are costed according to the amount of waiting and the total elapsed time for a given simulation. The smaller these two values, the fitter the Arbitrator. Hence the Arbitrators, initially randomly generated, are allowed to co-evolve with the plan organisms. Each individual's fitness is calculated according to its total cost. This means that selection pressure takes account of both optimization problems: interactions during phase two that increase an individuals cost will reduce its chances of reproduction, just as will a poor result from phase one of the costing. In general, a population of co-evolving Arbitrators could be used to resolve conflicts due to a number of different types of operational constraint.

The first implementation of this model, on a MIMD machine, had the various populations on separate processors and involved a complicated ranking mechanism to

allow co-evolution to produce useful results, global selection was employed. The second, more satisfactory, implementation spreads each population over a 2D toroidal grid. Selection is local very similar to the schemes used by Hillis, interactions are also local. The second phase of the costing involves individuals from each population at the same location on the grid. This provides a highly parallel model, which consistently provided better results faster than the first, less parallel, implementation. Experiments with up to 10 jobs have been conducted. Very promising results have been obtained for this extremely complex optimization problem, never before attempted. The search spaces involved are unimaginably huge, greater than 10^{100}, but this model has exploited parallelism sufficiently to produce good results. Machining costs dropped significantly across all the populations, as did the total elapsed time.

To date this system has used a rather simple job shop model where all the jobs are available at any instant and processing times are deterministic. The model is presently being developed to handle highly dynamic stochastic situations. Early results are very promising -- the fact that the populations hold a range of solutions can be exploited to great effect.

Variations of Genetic Algorithms

Simple Genetic Algorithm (Non-Overlapping Populations)

The simple genetic algorithm uses non-overlapping populations. In each generation, the entire population is replaced with new individuals. Typically the best individual is carried over from one generation to the next (this is referred to as elitism) so that the algorithm does not inadvertently forget the best that it found. Maintaining the best individual also causes the algorithm to converge more quickly;

in many selection algorithms, the best individual is more likely to be selected for mating.

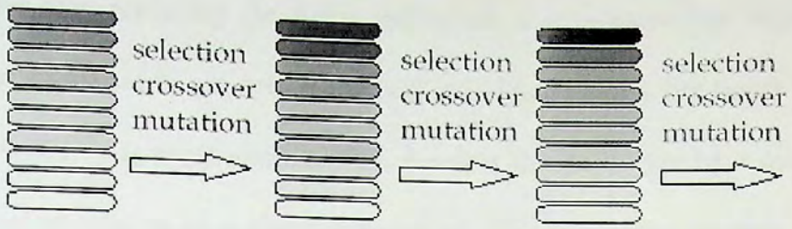

Figure 13: The simple genetic algorithm. This algorithm uses non-overlapping populations; the entire population is replaced each generation. Individual solutions are represented by shaded ovals. In this case, darker shading represents a better solution. [Wall 1996]

Steady-State Genetic Algorithm (Overlapping Populations)

The steady-state genetic algorithm uses overlapping populations. In each generation, the newly generated individuals replace a portion of the population. At one extreme, only one or two individuals may be replaced each generation (close to 100% overlap). At the other extreme, the steady-state algorithm becomes a simple genetic algorithm when the entire population is replaced (0% overlap).

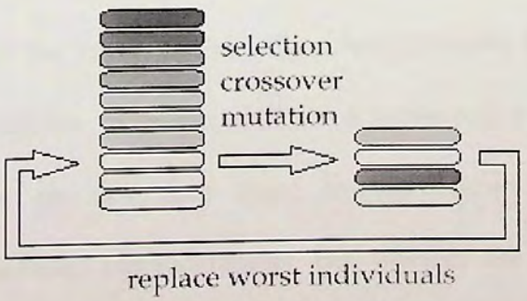

Figure 14: The steady-state genetic algorithm. This algorithm uses overlapping populations; only a portion of the population is replaced each generation. The amount of overlap (percentage of population that is replaced) may be specified when tuning the genetic algorithm.

Struggle Genetic Algorithm

The struggle genetic algorithm is similar to the steady-state genetic algorithm. However, rather than replacing the worst individual, a new individual replaces the individual most similar to it, but only if the new individual has a score better than that of the one to which it is most similar. This requires the definition of a measure of similarity (often referred to as a distance function). The similarity measure indicates how different two individuals are, either in terms of their actual structure (the genotype) or of their characteristics in the problem-space (the phenotype).

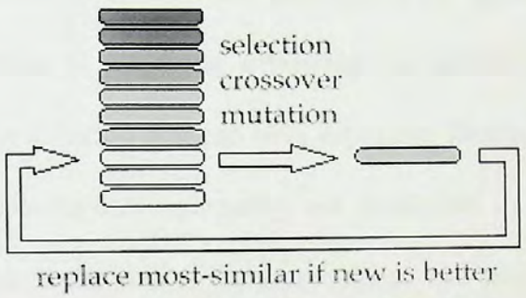

Figure 15: The struggle genetic algorithm. This algorithm is similar to the steady- state algorithm, but whereas the steady-state algorithm uses a replace worst strategy for inserting new individuals into the population, the struggle algorithm uses a form of "replace most similar". [Wall 1996]

Genetic Representation

Although much of the early genetic algorithm literature has focused on bit representations (i.e. solutions were encoded as a series of 1s and 0s), genetic algorithms can operate on any date type. In fact, most recent scheduling implementations use list-based representations. But whether the representation is a string of bits or a tree of instructions, any representation must have appropriate genetic operators defined for it. The representation determines the bounds of the search space, but the operators determine how the space can be traversed.

For any genetic algorithm, the representation should be a minimal, complete expression of a solution to the problem. A minimal representation contains only the information needed to represent a solution to the problem. A complete representation contains enough information to represent any solution to the problem. If a representation contains more information than is needed to uniquely identify solutions to the problem, the search space will be larger than necessary.

Whenever possible, the representation should not be able to represent infeasible solutions. If a genome can represent an infeasible solution, care must be taken in the objective function to give partial credit to the genome for its "good" genetic material while sufficiently penalizing it for being infeasible. In general, it is much more desirable to design a representation that can only represent feasible solutions so that the objective function measures only optimality, not feasibility. A representation that includes infeasibles increases the size of the search space and thus makes the search more difficult.

CHAPTER IV THE PCBS PRODUCTION SCHEDULING PROBLEM

Electronics industry is a major part of modern manufacturing, and electronic systems play an increasingly important role in a majority of today's products. Electronic systems are usually implemented with printed circuit boards (PCBs), and, consequently, PCB assembly has become an important sector of the electronics manufacturing industry overall. However, operating effectively in this industry is becoming more difficult as the companies must compete with high quality standards, rapidly changing technologies, short production cycles, and increasing product variety and complexity. In addition, the capital equipment cost of electronics assembly industry facilities is high in comparison to the usual turnover of a company. As a result, production planning decisions need to be made more and more frequently due to continuous changes in the production conditions. [Johnsson et al. 1999]

A common characteristic in the printed circuit board (PCB) assembly industry is that customers demand more functions and flexibility each year in the products they buy. In addition, consumers expect reliable and cheap products, and thus a common goal in the PCB assembly industry is to put more functions into a board with the same size and cost [Teng and Garimella 1998].

PCB assembly requires complete agility and reliability, which are only achievable with the use of robotics [Holcomb 1995]. Manual assembly methods may provide the needed flexibility, but they cannot provide the reliability and speed of robotic automation. When properly tooled, robotic assembly allows quick change from one product to another, handling a higher mix of products with reliability rates well in excess of non-robotic systems. PCB assembly is characterized by designs that range from simple and low-value board assemblies to very complex and high-value

board assemblies. Production volumes for different products vary in a very wide range (from millions to less than ten). One assembly system may encounter the assembly of PCBs with frequent design changes in small-batch production, whereas another system may assemble PCBs with a design that is fixed for six month or longer.

A recent development in PCB assembly is the growing role contract manufacturing. Many original equipment manufacturers (OEMs) have abandoned the assembly line in favor of outsourcing the manufacturing functions to contract manufacturers (CMs). CMs differ from OEMs that they build a variety of products for many different customers, whereas OEMs build only their own products. Despite the wider product variety and more dynamic product demand, CMs are expected to operate more efficiently than OEMs. This trend further emphasizes the importance of developing better production methods and systems.

Modern automated printing robots allow fast and high volume component insertion on Printed Circuit Boards. The efficiency of the machine lines is, however, highly dependent on the skill of the manufacturing engineers and the quality of the support systems. One has to solve several difficult interrelated controlling and scheduling tasks; the optimal use of different types of printing robots, the grouping of products to product families, the balancing of the work load of line machines, and the scheduling of the production of the whole plant

Fundamentals

A printed circuit board (PCB) is a substrate of a glass fabric impregnated with a resin (an organic polymer which, when mixed with a curing agent, cross-link to form a thermosetting plastics; usually epoxy). A PCB consists of one or more layers of

metal conductors and insulating material that allow for electronic components to be electronically interconnected and mechanically supported. A PCB of smaller dimension is commonly referred to as a card. A panel is an array of (usually identical) separate circuits fabricated on a single substrate.

The simplest form of PCB is the single-layer, single-sided board, which contains metalized conductor on one side of the board only. Greater levels of complexity and component density can be achieved by making double-sided and multi-layered boards: In double-sided assembly the PCB is assembled with components on both sides of the substrate, and multi-layering permits tracks to cross over one another, giving the designer more freedom in component layout.

Electronic components are either inserted through holes (e.g., griplet, axial and radial components) in the copper tracks and soldered in position, or are placed directly on to the surface of the board and soldered. These two distinctly different methods of manufacturing PCBs have given rise to different branches of manufacturing technology. The conventional method is known as through hole plated assembly, which is still popular for many applications, especially for low volume and manual assembly. Modern surface mount technology (SMT) utilizes smaller "flat" components, which are well suited to automated assembly process. They are common in small consumer products (e.g., cellular phones), whereas in the larger ones (e.g., televisions and computer monitors), where the competitive product price is a key factor, through-hole components are still widely used.

Components have also other properties that affect the assembly process. The size of the component defines the recognition camera type, the feeder size and the nozzle (or tool) which must be used when the component is handled. Furthermore,

component polarity, orientation in the input tapes and different handling speeds (e.g., pickup, recognition, placement and turret) affect the operation of the insertion machines.

Almost all machine types on the market operate in a similar fashion: The substrate is either placed (by the operator) or automatically transported to the staging area. After that, the components are "picked" from assigned pickup bin locations by vacuum and usually realigned either mechanically or optically before they are placed into the appropriate locations on the board.

Some machine types are flexible in the sense that they can handle a wide range of different substrate sizes as well as a wide range of different component types, whereas others are restricted to a condensed set of components, which they can operate at a much higher speed.

A feeder supplies the placement head with components in the proper orientation. Notwithstanding the machine type, the feeder capacity of the machine is usually expressed in the number of 8 mm tape feeders, which is used in almost all currently available machine types. Other feeder types include a stick of components, a vibratory slope feeder, and a tray feeder. [Johnsson et al. 1999] A component setup comprises the required operations to replace one tape feeder to another, and a machine setup comprises the required component setups, width adjustments, tooling plate changeovers and printing program updates to change manufacturing from one PCB type to another.

Wittrock presents the flexible flow line (FFL) environment that comprises several machine banks or production phases. In this model, the machines in a single machine bank are identical with each other, and a product can be processed in any

machine belonging to the machine bank, or it can skip the phase altogether. Each product passes the phases in a predefined order, and the transfer between stages is accomplished with the help of magazines or some other form of transport. The setup time between different products, which is assumed to be negligible, is ignored. Hence, the processing time is a function of the processed product and the phase. [Wittrock 1988]

Johnsson et al. introduce in a generalized flexible flow line (GFFL) environment, which is a generalization of the FFL. [Johnsson et al. 1997] The GFFL environment also comprises successive machine banks, but the type of the machines can vary even inside a particular machine bank (unlike in FFL). The machine type defines the speed of the machine, and thus the processing time in GFFL is a function of the product and the machine type. Moreover, setup times are also taken into account (unlike in FFL).

The PCBs are transferred from one phase to another either manually (e.g., in magazines which can hold 10-100 PCBs) or with a conveyor belt. The conveyor-linked machines are usually referred as being coupled, and a system with a batch transfer as uncoupled. Because the change of board type causes a setup, boards of the same type are collected in a batch, in which they are operated successively. The batch size can vary considerably: A mass-product PCB may remain in active production for several months,

PCB Assembly Classifications

Johnsson classifies in [Johnsson 1999] the literature on PCB assembly according to the number of different PCB types and machines present in the problem. Accordingly, the four main problem classes are:

One PCB type and one machine (1-1) class comprises single machine optimization problems, where the goal is to minimize the printing time of the machine. The class can be further divided into four subclasses:

- feeder arrangement problems (1-1a),
- placement sequencing (or insertion order) problems (1-1b),
- nozzle assignment problems (1-1c), and
- component retrieval problems (1-1d).

Multiple PCB types and one machine (M-1) class comprises setup strategies for single machine, where the goal is to minimize the setup time of the machine. The setup strategies can be classified as follows:

- minimum setup strategy (M-1a),
- group setup strategy (M-1b), and
- partial setup strategy (M-1c).

One PCB type and multiple machines (1-M) class concentrates on component allocation to similar machines, where the usual objective is balancing the workload of the machines in the same line (usually by eliminating bottlenecks).

Multiple PCB types and multiple machines (M-M) class or scheduling problems usually concentrate on

- allocating jobs to lines (M-Ma) which includes routing, lot sizing and workload balancing between lines, and
- line sequencing (M-Mb).

Related Work for Class (1-M)

In this thesis we are only considering class (1-M). Only few papers considering the case of similar (sequential) machines in the same production line have been put forth. Here, the most eminent criterion is workload balancing so that the bottlenecks of the line are eliminated.

Lofgren and McGinnis present a soft configuration decision, which has an impact on two key criteria: workload on each machine (i.e., balancing), and material handling (i.e., machine visits). The soft configuration problem specifies the attributes, which are available on each machine, and, therefore, it determines the set of operations that could be performed. The authors consider three operating policies: dynamic (where tools are added or removed so that the required operation can be done in one machine without routing it to another), static (which specifies configuration for a finite production horizon and routes jobs to appropriate machines) and pseudo-dynamic (where some attributes are static and the rest dynamic). The authors give heuristic algorithms for static and dynamic operating policy, where the objective is to maximize machine utilization and minimize material flow transactions. [Lofgren and McGinnis 1986]

Ben-Arieh and Dror consider assigning components in a case of two insertion machines, so that all boards in a production plan can be produced, and the output rate is maximized. They give a mathematical formulation of the problem and solve it with a heuristic algorithm. [Ben-Arieh and Dror 1990]

Askin et al. discuss a surface mount technology plant with multiple identical machines. They present a four-stage approach for grouping the boards and allocating the components to the machines: First, the boards are grouped into production

families. Next, for each family, the component types are allocated to the machines. After that, the families are divided into board groups with similar processing times. Finally, the groups are scheduled. The objective is to minimize the makespan for assembling a batch of boards and to reduce the mean flow time. The authors present and compare three heuristic methods - component-assignment/workload balancing algorithm (CAWB), workload balancing algorithm with shortest total processing time (WBASPT), and natural board subfamily algorithm (NBSA) - and conclude that CAWB and WBASPT outperform NBSA. [Askin et al. 1994]

Watkins and Cochran consider a line of similar insertion machines, and propose a heuristic method for re-balancing the workload by moving components from the bottleneck machine to other machines. However, each move is associated with a cost, and the method finishes when the cost of a move outweighs savings. [Watkins and Cochran 1995]

McGinnis et al. give a mathematical model for component allocation for both coupled and uncoupled machines. [McGinnis et al. 1992] Ammons et al. continue the work by considering component allocation to two or more placement machines, when the objective is to balance a combination of the assembly time and the machine setup time. When machines are coupled, the workload balancing reduces to maximizing the throughput of the bottleneck machine on line. The authors approach the component allocation problem by developing two heuristic methods based on list processing and branch-and-bound technique. Furthermore, they give an integer programming formulation, which tries to minimize the maximum combined PCB assembly and machine setup time for each PCB over all machines. [Ammons et al. 1997]

Brandeau and coworkers consider assigning components to machines in an assembly plant with different types of work phases (automatic, semi-automatic and manual). The goal is to minimize the total setup and processing cost for assembling all boards. Brandeau and Billington present two heuristic algorithms: stingy component (which tries to avoid assigning less frequently used components to the automatic work phase) and greedy board (which tries to assign a whole board to a single work phase instead of splitting it). After a set of tests, the authors conclude greedy board to be a better method of the two, because the setup cost are high relative to the insertion costs. [Brandeau and Billington 1991] Hillier and Brandeau extend the same problem by presenting a new mathematical model and an improved heuristic based on branch and bound technique. [Hillier and Brandeau 1998] The same authors further expand the mathematical model by introducing a workload balancing criterion and introducing CMWB (cost minimizing, workload balancing) heuristic.

Commercial Software Systems

There are several software tools for production planning available in the market but relatively few of them are specialized in the PCB assembly. Generic tools seldom suit for production planning in PCB manufacturing due to the special requirements associated with it: Firstly, the system must have knowledge about the machinery used in the production environment, its capabilities and limitations. Secondly, a production planning tool should have a tight integration with the existing systems, such as CAD/CAM (computer aided design/manufacturing) or MRP (material resource planning) systems. Thirdly, in practice the usability of a system reduces drastically, if it does not produce NC-codes for the machines. Since PCB manufacturing facilities

are usually highly automated, the goal of a production planning system is not to impose any further work to the personnel. [Smed et al. 1999]

Some of the most popular software systems designed to support PCB assembly and their key features are listed in Table 3. Concentration is on the optimization features, albeit optimization is rarely the primary task in these systems. The systems are mainly designed for automating the PCB design and manufacturing processes, such as joining bill of materials (BOM) files and CAD files together and generating NC-codes for machines. The information is gathered from the vendors' web pages and brochures. The list of the machine vendors' systems includes only a few entries although almost all machine types in the market have their own proprietary system. However, the current trend is that the machine vendors are starting to co-operate with the general system providers.

In Table 3 the systems are divided into two categories: tools provided by the machine vendors and tools provided by independent companies. A machine vendor's system usually supports only its own equipment and includes a concise interface and some simple code optimization routines. Conversely, since production plants may comprise multiple machines from different vendors, the independent systems aim at providing better support (e.g., a centralized component library, better CAD support, product transportation from one machine to another) for various machine types and makes. Some systems also contain networking capabilities that allow them to integrate into a local-area network. These systems can obtain on-line data from the machines and use it to control the production. The word `optimization' is used here loosely, since in some systems the term "optimal solution" actually means a feasible solution. Features of the systems vary greatly. Some use a hierarchical decomposition strategy to optimize each level of production ranging from the printing order and

	General Information			Optimization				
	Vendor	Package	Plateform(s)	Printing Order	Feeders	Setup	Line Balancing	Scheduling
Independent Companies	Dareba Corporation	Cim-Scan	DOS, Windows, UNIX	√	√	√		√
	Mitrom Corporation	CIMBridge 97	Windows	√	√	√	√	√
	Tecnomatix-UNICAM Software Inc.	UNICAM	Windows	√			√	
		Factory Advisor	Windows			√	√	√
	Tecnomatix Technologies Ltd.	EXALINE	Windows, UNIX			√	√	
	Unisoft Corporation	EZ-CIM/EZ-CAM	Windows	√			√	
	FABMaster Inc.	FABMaster	Windows	√	√	√	√	√
	AIS Corporation	CircuitCAM	Windows	√				
	Optelco	Optel	Windows	√	√	√	√	√
	Harelda Oy.	SMDOPTIMIZER	Windows	√	√	√	√	√
Machine Vendors	Universal Instrument Corporation	UIC	Windows, OS/2	√	√			
	Fuji America Corporation	FujiCam	Windows	√			√	
		XCME	Windows	√				
	Panasonic Factory Automation	PanaCIM	Unix	√	√	√	√	
		PanaPro	Windows	√	√	√	√	
	Sony	TIMMS	OS/2	√	√	√	√	√

Table 3: Summary of the optimization features of commercial PCB assembly support systems. [Smed et al. 1999]

feeder arrangement optimization of a single machine to the scheduling of the entire production environment. Others offer only a few means to improve the operation of a machine or balance the component allocation between machines. The features listed in Table 3 are printing order and feeder optimization of the supported machines (1-1), setup strategy optimization (M-1), balancing the load between machines in a line (1-M), and scheduling the jobs, which includes their allocation to the lines (M-M). Since in many cases it is difficult to estimate the quality of the solutions produced by the systems, we have not evaluated the utility of the optimization algorithms used in the systems in greater detail. [Smed et al. 1999]

Problem Definition

A flow shop system includes a number of phases and each product runs through the phases in the same phase-to-phase order. The usual case is one machine per phase with an obligatory visit of each phase. A special case of flow shop scheduling presupposes that the operating sequences of the jobs are the same on every machine. [Kim 1996] has given heuristic algorithms to this problem, called permutation schedule flow shop. Single machine per phase with a possible skip of some phases (Flexible Flow) has been discussed in [Shmoys 1994]. This variant is a special case of job shop scheduling where the order of visiting the machines (i.e. phases) depends on the product. [Johnsson et al.1997]

Network Flow Shop and Hybrid Flow Shop are generalizations of flow shop where each phase may include several identical machines. Setup times or machine lines are not considered in this problem formulation. In Flexible Flow Line (FFL) the machines in each phase are identical and the processing time of a product is thus a function of the product and the phase. Any of the machines of a certain phase can be

used for a product. Set-up times are neglected in this model and transportation is performed (without batches formed by full magazines) by an automated mechanism. Wittrock solves the FFL-scheduling problem for the PC-board production problem by dividing the problem into three sub-problems.

Machine allocation maps the products to the machines at each phase. Sequencing fixes the order of the products for each machine. The last step, scheduling, determines the starting and terminating moments of the processing of the products on the machines. In Wittrock's case the magazines were of the same size and a product was thus considered as a full magazine. Thus the processing of a job must be completed at the previous phase before the starting at the next phase. Setup times are not important in FFL.

The makespan (C_{max}) is commonly used as an optimization criterion in FFL and flow shop models. Another property that is typical of flow shop problems is that they usually presuppose permutation schedules, i.e. the sequence of the jobs remains the same for all machines.

Input Data

In this work we are concerned mainly with the line balancing part of the PCBs assembly line production scheduling. This sub-section describes the needed input data.

A job stands for one product to be produced with a defined quantity within a defined time. The following points denote the required input to the scheduling system for a job:

- Layout description of the PCB to be produced. Layout description includes the entire data for each component to be mounted on the board.
- Number of PCBs to be produced for the job.
- Delivery date for the job.
- Possible starting date for the job.
- Priority of the job.

The production line layout provides information about the machines order and types. The following points denote the required input to the scheduling system for a production line:

- Machine types available in the production line.
- Position of each machine in the production line.
- Status of each machine in the production line.
- Fixed feeder settings of each machine.
- Current feeder settings of each machine.

The feeder holds the components types and make them available for the production machines. The following points denote the required input to the scheduling system for feeders:

- Feeder specific time to add a new feeder.
- Feeder specific time to remove a feeder.
- Feeder specific time to change the component type put on a feeder.

Each machine type differs from the other in the component types that it can handle. The following points denote the required input to the scheduling system for machine types:

- Component types that can be handled by each machine type.
- Average machine dependent insertion times.
- Average number of available magazines, depending on the component type.

Fundamental Steps

The following tasks should be performed in order to optimize the production process:

- Assignment of jobs to different production lines and clustering.
- Distribution of the components of the PCBs in a cluster to the different production machines in a line (Line-Balancing).
- Optimization of the time to change the feeders.
- Optimization of the time to insert the components.

The jobs should be assigned to the different production lines, so that each line will do an equal amount of work. The clustering is provided to collect jobs, for which the same feeder setup shall be used. This means, there shall be no change of the feeders during the production of one cluster of jobs.

The following constraints must be followed: -

- If there are jobs, that require a special line for production, they have to be placed on it. A reason for this can be a PCB that has a very big size or a component on a board that can be placed only by a machine of this line.

- Dependent on the importance of the jobs, priorities can be assigned, so those jobs with high priorities should be scheduled first.
- A check should be performed for the number of component types that are used to produce the job and the ones that are already placed on each line (standard or from the last job). Then the job shall be assigned to that line that contains most of the required component types.

An average expenditure estimation, that is necessary to produce a job, can be calculated in the following way:

Number of components per board: C_b

Number of boards per job: B_j

Average number of components per hour that can be inserted on the line: C_{lh}

Relative average expenditure: $C_b * B_j$

Absolute average expenditure: $T_{jl} = (C_b * B_j) / C_{lh}$

The average number of components per hour that can be inserted on the line can be calculated by adding the average numbers of components per hour $C_{m(i)h}$ that can be inserted by each production machine of the line:

$$C_{lh} = C_{m(1)h} + C_{m(2)h} + C_{m(3)h} + \ldots + C_{m(m)h}$$

The duration taken by the production line is:

Maximum $(T_{m(1)}, T_{m(2)}, \ldots, T_{m(m)})$ where $T_{m(i)}$ is the time taken by the i^{th} machine in the production line.

The following algorithm distributes the jobs to the lines and gives a sequence for production:

1) Sort the jobs according to the following rules:

   a) Priority (high priority first).

   b) Number of lines that can handle the job (one possible line first).

   c) Relative average expenditure (big jobs first).

2) Calculate the current work for each line. This is the sum of the absolute average expenditure of all the jobs assigned to that line.

3) Assign the next job in the sorted list to the line that currently has the least work and can produce that job.

4) Go to step number two unless all jobs are assigned.

The following algorithm builds clusters: -

1) For every line, an average number of magazines have to be defined; this means the average number of different components that can be set up on the different machines of a line. This can be calculated by adding average numbers of magazines of each machine.

2) To build clusters for a line, the algorithm should count the number of component types of the first job that is assigned to this line add the number of different component types from the second job and so on, until the sum reaches a value of about 95% (should be user-defined) of the average number of magazines of the line. These jobs will form the first cluster.

3) Step number two shall be repeated for the next clusters until all jobs are assigned to a cluster.

To calculate the complete time, that is used to produce a cluster, not only the insertion times for the clusters have to be calculated, but also the time to change the feeders between the production of two clusters. For this average time can be defined:

- Feeder specific time to add a new feeder to the magazine area.
- Feeder specific time to remove a feeder from the magazine area.
- Feeder specific time to change the component type put on a feeder.

The system should check the number of feeder changes that are necessary to switch from one cluster to the next and add the times. These times will be added to the insertion time.

When clusters are moved on a line or between different lines, this time value will change, because it depends on the settings done for the previous cluster.

A check, whether the magazines available for a line are full, should be performed, before a job is assigned to a line. This can be done by adding the average numbers of available magazines on each machine of the line, dependent on the component type. If the number of different component types assigned to a machine reaches this limit, this line shall not be used for further assignments.

The main task of line balancing is the distribution of the component types required for the clusters to the machines of a line. Because there should be no change of the feeder settings during the handling of a cluster, the component types of all the jobs of the cluster are merged together.

The assignment of the component types to the machines requires a special order: -

- The component types should be sorted according to the number of machines that can handle these types. Components, which can be handled by only one machine or very few machines, have to be assigned first.

- For each component type the number of components required for each job in the cluster has to be estimated. The component types have to be sorted according to the maximum values.

- The components can only be assigned to machines, which can handle that type. Especially for components, that can only be handled by a special machine, will be assigned to it and not changed any more.

- Standard assignment; if a component type is already set up on a machine, which means, that it can be either a standard setting or a setting from the previous cluster, then the component shall be assigned to this machine again.

A check, whether the magazine areas for the machines are full, should be performed, before a component type is assigned. Defining an average number of available magazines, in other words, an average number of different component types that are usually handled on the machine, can do this. If the number of components assigned to a machine reaches this limit, this machine shall not be used for further assignments.

The optimization of the feeder setup and the insertion sequence for a machine depends essentially on the type and the construction of the machine. For some machines, it is even not possible to divide the two-optimization steps from each other, which means, that the organization of the feeders depends directly on the insertion sequence.

To optimize the insertion time, the insertion path has to be minimized. Because the machine picks up a component from a feeder, before it places it on the board, it is not sufficient to create the shortest path between the components of a board. The movements to and from the feeders have to be taken into consideration also. These movements can be minimized, if a feeder is placed nearby the locations on the board, where the components are to be placed, that are taken from this feeder.

To provide this optimization, it is necessary to describe the complete geometry of the workspace of the machine. It is not enough to use the description of the board, also the place, where the board will be load into the machine and the location of the feeder areas has to be described. This information has to be added to each machine.

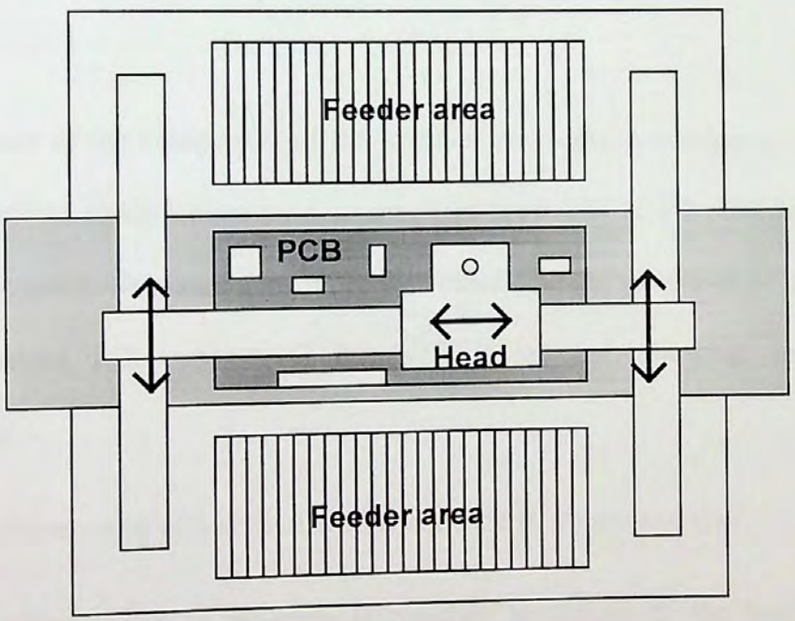

Figure 16: PCB production machine

The optimization also depends on additional movements of the head that are necessary to insert a component, e.g. to perform optical correction. The place of additional stations has to be described. Standard assignments and current assignments should be taken into account.

CHAPTER V GAPS: SOLUTION METHOD

GAPS is only addressing the "line balancing" problem out of the different steps needed for a complete PCB assembly production scheduling as illustrated in chapter IV. GAPS aims at minimizing the makespan of the PCBs production by selecting component type assignment that introduces minimum idle time for each machine and accordingly minimize the total time for the production. This type of problem is similar to the flow shop problems but it has an extra constraint. The extra constraint is that each component type can be handled by more than one machine. Accordingly this extra constraint adds extra difficulty to the problem. A machine must be selected for each component type that will lead to the minimum idle time for each machine in the production line.

Because of the complexity of this class of problems, a number of simplifying assumptions has always been used in practical applications. [Husbands 1994] The assumptions made when modeling a problem determine the variations of that problem that the model will support. GAPS is built on the following simplification assumptions: -

- The machine speed of insertion is constant for any component type.
- The machine speed of insertion is constant regardless of the location of the component on the board.
- The machine has infinite number of magazines to hold different component types.
- No components are placed on the machine feeders.

Main Modules

GAPS is mainly divided into the following modules: -

- Input Module.
- Initialization Module.
- Genome Interpretation Module.
- Evaluation Module.
- Population Generation Module.

Each module is designed and implemented separately. This architecture facilitates the replacement of any module as well as smooth the process of its testing.

Input Module

The input module is responsible for getting the needed data for the job, production line, processing time, due dates, and precedence constraints.

The following is the needed job data: -

- PCB name.
- The number of PCBs that should be produced.
- The layout of the PCB i.e. the various components that should be inserted.
- The precedence constraints that should be applied.
- The possible starting date.
- The due date.

The following is the needed production line data: -

- The machines that are placed in the production line.
- The order of the machines.
- The availability of each machine.

The following is the needed machine data: -

- Machine average insertion time.
- Number of magazines that it can hold.
- Machine type.

Initialization Module

The initialization module is mainly responsible for initializing the population with feasible schedules. It is divided into two steps:

- Line assignment.
- Population initialization.

The line assignment algorithm that is implemented in GAPS is the one outlined in chapter IV.

The following algorithm is used to initialize the population with feasible schedules that will be the input to the population generation module: -

- Create a list of the different component types in each job. The precedence constraints are taken into consideration.
- For each component type a list is created for all the machines in the production line that can handle it.

- Each genome in the population is filled by randomly assigning to each component type a machine. The machine is selected from the list created in the previous step.

Genome Interpretation Module

The genome interpretation module provides the timing for the operations to start. In other words in converts the genome to an actual schedule with start and end time for each operation. The output of this module is the input to the evaluation module.

The output of this module is a list of machine names that are assigned component types. For each machine there is a list of all the component types assigned to it as well as the number of each component type.

The following is an example of a genome after it has been converted to a list of assignment per machine from which a fitness value can be evaluated: -

```
Total Duration: 1308
(Machine <ENDMON>: Duration 1308
V26803-B73-V2 (104) V26801-B76-V77 (42) V26803-B12-V51 (33) V26801-
B76-V85 (17) V26801-B76-V53 (16) V26802-B14-V3 (15) V26801-B76-V61
(13) V26803-B12-V9 (13) V26801-B71-V49 (7) V26801-B76-V109 (6)
V26801-B76-V89 (6) V26803-B12-V31 (6) V26801-B76-V13 (5) V26813-B356-
V7 (4) V26801-B76-V200 (4) V26803-B75-V2 (4) V26801-B71-V34 (3)
V26812-B37-V10 (3) V26801-B71-V45 (3) V26802-B14-V2 (2) V26808-B6829-
V41 (2) V26802-B14-V25 (2) V26808-B1049-V40 (2) V26811-B144-V40 (2)
V26801-B76-V93 (1) V26801-B71-V216 (1) V26808-B7137-V40 (1) V26811-
B69-V40 (1) V26810-B306-V40 (1) V26811-B350-V40 (1) V26811-B297-V40
(1) V26808-B1071-V40 (1) V26808-B2714-V40 (1) V26808-B1039-V40 (1)
V26808-B6197-V40 (1) V26801-B76-V21 (1) BOHR152 (1))
(Machine <HANDMON>: Duration 210
V26898-B597-V1 (1) V26810-B397-V50 (3) V26827-B457-V40 (2) V26817-
B89-V40 (1) PAD92-K (10) CHIP-PASS (7) BZA100 (3) V26815-B203-V1 (1)
V26827-B469-V15 (2) V26827-B431-V30 (1) V26827-B469-V8 (1) V26827-
B392-V7 (1) V26827-B470-V106 (1) V26803-B31-V31 (4) V26827-B214-V5
(2) V26827-B431-V16 (1) V26827-B431-V34 (1))
(Machine <LOETROB>: Duration 332
PAD32 (103) PAD64 (50) GNDBOHR4.3 (5) V26827-B442-V8 (1) V26823-B7-V7
(1) V26898-B540-V300/V26898-B540-V3 (1) V26827-B478-V8 (1) V26827-
B454-V160 (1) V26827-B392-V1 (1) V26827-B123-V5 (1) V26827-B163-V2
(1))
(Machine <SIEMENS>: Duration 1308
V26810-B301-V50 (9) V26808-B6356-V40 (1) V26808-B7151-V30 (1) V26808-
B7176-V30 (1) V26808-B7030-V30 (1) V26808-B7205-V30 (1) V26121-B49-
V50 (1) V26803-B73-V1 (50) V26813-B356-V3 (38) V26803-B54-V101 (26)
```

```
V26803-B12-V50 (15) V26813-B356-V1 (14) V26802-B14-V17 (9) V26801-
B76-V81 (7) V26801-B76-V29 (6) V26801-B76-V33 (6) V26803-B12-V10 (4)
V26801-B71-V85 (4) V26801-B76-V69 (4) V26808-B1059-V40 (3) V26808-
B4168-V40 (2) V26802-B14-V23 (2) V26811-B293-V40 (1) V26811-B148-V41
(1) V26801-B76-V57 (1) V26801-B71-V235 (1) V26801-B71-V268 (1)
V26801-B71-V251 (1) V26801-B76-V41 (1) V26801-B71-V240 (1) V26802-
B14-V5 (1) V26808-B1041-V40 (1) V26808-B7136-V30 (1) V26808-B2403-V40
(1) V26810-B304-V40 (1))
(Machine <SIPLACE>: Duration 1302 V26808-B7175-V20 (1) V26121-B266-
V50 (4) V26823-B5-V1 (2) V26808-B6717-V50 (2) V26808-B7204-V30 (1)
V26823-B6-V3 (1) V26808-B7169-V35 (1) V26808-B6842-V31 (1) V26801-
B71-V200 (44) V26803-B12-V26 (33) V26801-B76-V25 (22) V26803-B12-V19
(15) V26802-B14-V8 (13) V26801-B71-V37 (9) V26802-B14-V1 (6) V26803-
B12-V11 (5) V26802-B14-V7 (4) V26803-B54-V9 (4) V26803-B12-V21 (3)
V26808-B2505-V40 (2) V26801-B71-V121 (2) V26801-B76-V101 (2) V26808-
B1043-V40 (2) V26801-B71-V226 (1) V26801-B71-V229 (1) V26811-B34-V40
(1) V26808-B1042-V40 (1) V26802-B14-V9 (1) V26803-B12-V15 (1) V26803-
B12-V27 (1))
```

Evaluation Module

The evaluation module estimates the performance of each schedule generated. The time needed by each machine in the production line is calculated. The machine needing the maximum time is the one that determines the production time needed for the job. The evaluation module is one of the main parts of GAPS cause it guides the search algorithm.

The following is an example for a production line. The duration for each machine is calculated. Then the production time needed is obtained from the machine having the maximum duration.

```
Duration: 1308
(Machine <BEV_SATZ>: Duration 3)
(Machine <ENDMON>: Duration 1308)
(Machine <HANDMON>: Duration 210)
(Machine <LOETROB>: Duration 332)
(Machine <Royonic 410>: Duration 0)
(Machine <RUNDLAUF>: Duration 36)
(Machine <SIEMENS>: Duration 1308)
(Machine <SIPLACE>: Duration 1302)
```

```
(Machine <SMDMON>: Duration 30)
(Machine <THT_AUT>: Duration 10)
(Machine <VORMON>: Duration 162)
```

Population Generation Module

An order-based representation for each component type is used in the implementation. The length of the genome depends on the number of different component types to be assembled on the PCB. Each gene consists of a component type code, machine code and the number of components to be inserted of this type.

Component type 1	Component type 2	Component type 3	...
Machine code	Machine code	Machine code	...
Number of components	Number of components	Number of components	...

GAPS system uses the following three crossover operators: -

- Single point crossover:

The offspring chromosome is initially empty. Then the genes are copied from the first parent till the cut point, which is generated at random. The copied genes are deleted from the second parent. The remaining genes in the second parent are then copied to the new offspring.

- Double point crossover:

This operator is similar to the single point crossover but instead of having a single point cut there are two cut points in the first parent. The genes located between the two cut points are transferred to the offspring chromosome. These genes are deleted from the second parent. Then the remaining genes in the second parent are copied to the offspring in the same order.

- Generalized Uniform crossover (GUX):

In this operator the offspring chromosome is initially empty. A parent is chosen at random and the operation at the first position of the parental chromosome is appended to the offspring. Then this operation is deleted from both parents. This step is repeated until both parent strings are empty and the offspring contains all the operation involved. [Mattfield 1995]

GAPS mutation operator reassigns all the component types to randomly picked machines. The termination criteria used in GAPS in a static number of generations defaulted to 100.

Figure 17 shows a population mapped onto a torodial-connected grid. A torus has the advantage of introducing a spatial distance between individuals by avoiding border locations. Each individual has four neighbors located to its east, north, west and south. This spatial structure can be seen as an artificial habitat in which mating is restricted to overlapping neighborhoods. A population structure as well as a suitable definition of the local mating scheme has to be chosen in a way such that a sufficient gene flow through the population and a sufficient spatial distance between the individuals is "well balanced".

Each individual has four neighbors and locally mates within this sub-population. The sub-population size of five individuals is too small in order to rely on selection holding the sub-population at regions of high fitness in the search space. Instead, the crossover rate is set to 1.0.

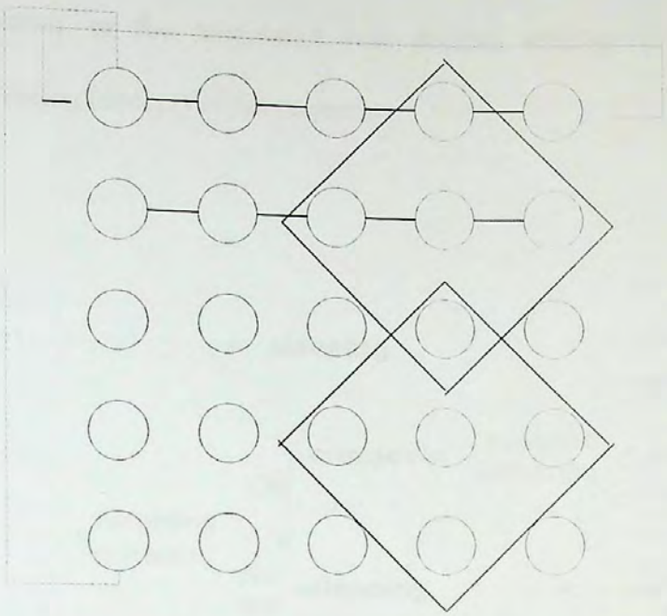

Figure 17: A Structured population on a toroidal grid avoids border locations. Overlapping sub-populations consists of five individuals each [Mattfield 1995]

In Figure 18 an individual first compares its fitness with the fitness in its neighborhood. If the fitness is superior to all neighbors, the conservative behavior will cause the individual to sleep. If several best individuals exist in one neighborhood none of them will be superior. For this reason incorporating attitude inheritance does not introduce an elitist strategy. [Mattfield 1995]

An inferior individual determines its attitude. Therefore its actual behavior is drawn probabilistically from an interval [0,1]. Initially a threshold is set to 1.0 enforcing crossover. Decreasing the threshold increases the probability of mutation. In case of crossover, the Hamming distance to the selected mate is evaluated. If mates differ in less than 1% of their genes it seems not worthwhile to try to crossover. Again the individual sleeps, but now because of a different reason. If crossover or mutation is carried out, the fitness of the offspring is evaluated. Either the offspring dominates

both parents (improve), or the acceptance rule decides whether to replace the individual by its offspring (accept) or not (reject).

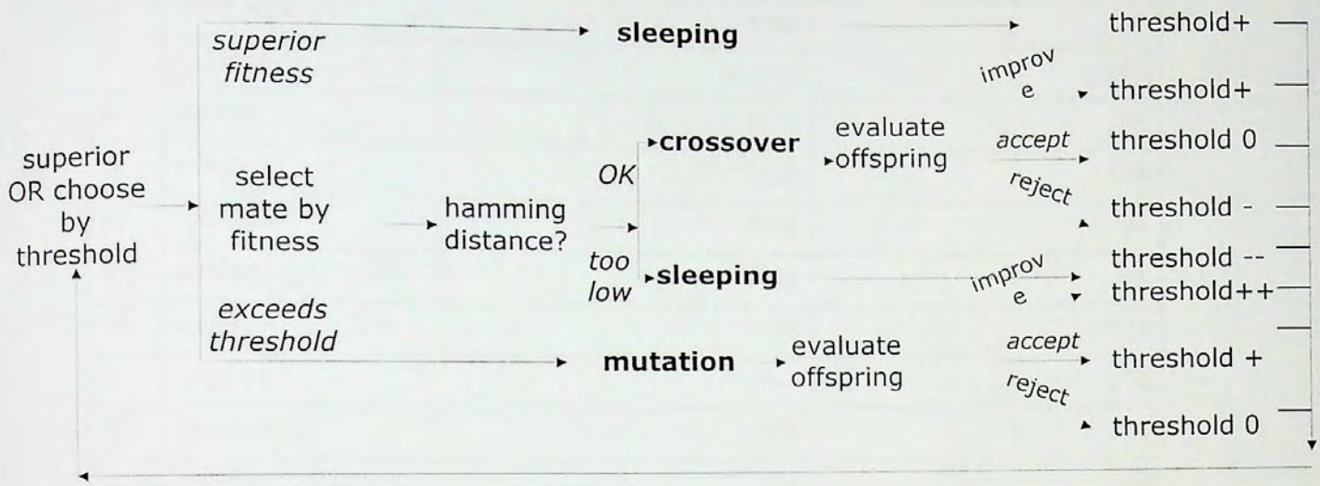

Figure 18: Control model of local recombination. [Mattfield 1995]

Test Cases

GAPS was tested on several number of PCBs varying from small to large-size boards. Number of components on the boards varied from 100s to 1000s. On the other hand several types of production lines were used in the different runs to determine GAPS performance. GAPS results could not be compared against the results of other systems cause the data was classified as confidential by the company's developing the systems. The various companies were not willing to reveal any portion of their data or their algorithms. Accordingly, the three variations of genetic algorithms explained in chapter III were developed in order to compare their results against the results obtained from GAPS.

	Board Name	Number of Components	Number of Part Types
1.	Fmr5r4k	13127	145
2.	VBPCBRWD	6891	165
3.	EEPD	5120	123
4.	d943_wg2	4462	150
5.	8121346	3756	93
6.	420285	3438	35
7.	47847	3330	148
8.	08204800	3244	94
9.	837239	2883	98
10.	COPR54	2486	224
11.	AMU41	1841	172
12.	D1123-W50	1778	266
13.	XSP8U	1734	63
14.	PSM01_C	958	91
15.	D665_2	647	75
16.	P732-D4.	523	142
17.	P730-E5	458	143
18.	P730-H8	457	127
19.	P230A-02	286	92

Table 4: Lists all the boards that are used in testing GAPS.

Table 4 gives a list of all the various boards used in testing the genetic algorithms. The first column gives the name of the board. The second column lists the number of components to be placed on each board. The third column gives the number of different part types that are on the board. More than one component usually has the same part type.

The following table gives a summary of the settings of the genetic algorithm used for the simple GA, steady state GA, struggle GA and GAPS.

Parameter	Description	Simple GA	Steady State GA	Struggle GA	GAPS
Representation	A triple array chromosome each gene contains component type, machine code and number of components.	Same	Same	Same	Same
Fitness evaluation	The fitness is evaluated by computing the makespan C_{max} for a decoded schedule.	C_{max}	C_{max}	C_{max}	C_{max}
Crossover op.	Single point, double point and generalized uniform crossover operators.	Same	Same	Same	Same
Mutation op.	Randomly changing the machine codes in the gene.	Same	Same	Same	Same
Crossover rate	The probability p_c for an individual to perform crossover	0.8	0.8	0.8	Auto adaptive
Mutation rate	The probability p_m for an individual to perform mutation.	0.2	0.2	0.2	Auto adaptive
Population size	A fixed number of individuals form the GA population.	100	100	100	100
Population structure	The individuals reside on a torodial grid.	10x10	10x10	10x10	10x10
# of neighbors	The number of individuals on which selection is based and from which a mating partner is chosen.	100	100	100	4

Selection scheme	The scheme used for the selection of a mating individual.	Roulette wheel	Roulette wheel	Roulette wheel	Fitness within the neighbor-hood.
Termination criteria	A fixed number of generations is used.	100	100	100	100

Table 5: Summary of the settings of parameters used for the various genetic algorithms.

In addition to the genetic algorithms that are implemented an exhaustive search methods is also implemented in order to be able to obtain the optimal solution and compare the results of GAPs against the optimal solutions.

GAPS Performance

Gen.	Simple GA			Steady State GA			Struggle GA			GAPS		
	Best	Ave.	Worst	Best	Ave.	Worst	Best	Ave.	Worst	Best	Ave.	Worst
0	232	305	504	232	316	465	232	305	504	232	305	504
10	228	311	468	232	253	264	228	317	468	224	275	440
20	232	306	440	228	244	252	236	312	476	220	265	432
30	232	312	452	228	244	252	228	324	468	220	277	416
40	224	302	452	228	244	252	224	312	428	220	270	468
50	236	314	436	228	244	252	228	316	452	220	261	428
60	228	305	432	228	244	252	228	311	456	220	266	416
70	228	312	444	228	241	250	232	309	476	220	275	456
80	228	300	468	228	241	250	236	309	468	220	272	444
90	224	294	410	228	241	250	228	316	448	220	268	440
100	232	317	460	228	241	250	232	314	472	220	267	484

Table 6: Lists the best, average and worst fitness obtained in each of the various genetic algorithms. The single point crossover operator is used. Board is called P730-e5. The optimal solution obtained from exhaustive search is 220 seconds. All the values in the table are in seconds.

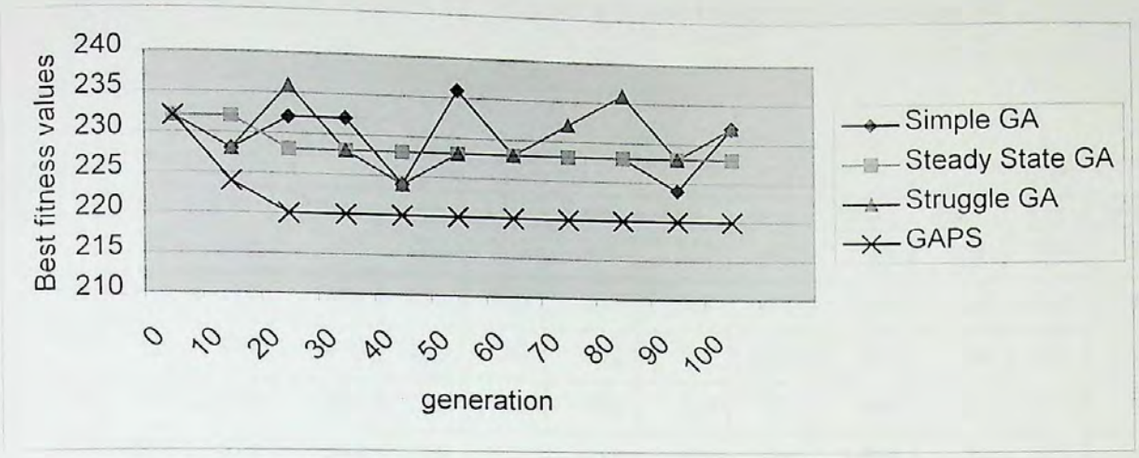

Figure 19: Compares the best fitness values obtained in each generation for the various genetic algorithms using the single point crossover operator on PCB P730-e5.

Table 6 lists the best, average and worst fitness obtained in each of the various genetic algorithms. The single point crossover operator is used. PCB is called P730-e5. The optimal solution obtained from exhaustive search is 220 seconds. All the values in the table are in seconds. As can be seen in table 6 the GAPS method was the only one that achieved the optimal solution even quite early in the generations. The steady state genetic algorithm converged too quickly so it got stuck in a local optima. This is due to the fact that all the worst individuals are removed each generation. On the other hand the performance of the simple genetic algorithm and the struggle genetic algorithm was quite reasonable despite that they did not reach the optimal solution.

Table 7 lists the best, average and worst fitness obtained in each of the various genetic algorithms. The double point crossover operator is used. PCB is called P730-e5. The optimal solution obtained from exhaustive search is 220 seconds. All the values in the table are in seconds. The performance of the Simple algorithm is almost similar to the previous results obtained when the single point crossover operator was

selected. There is a slight improvement in the Steady State algorithm and the Struggle algorithm was able to achieve the optimal solution in generation number 30.

Gen.	Simple GA			Steady State GA			Struggle GA			GAPS		
	Best	Ave.	Worst	Best	Ave.	Worst	Best	Ave.	Worst	Best	Ave.	Worst
0	232	316	480	232	320	495	232	309	465	236	309	575
10	228	309	474	232	243	252	228	312	416	220	286	408
20	228	305	456	232	243	252	236	297	432	220	283	415
30	224	309	424	232	243	252	220	311	465	220	273	400
40	228	315	468	228	241	250	232	321	515	220	277	428
50	224	315	480	228	241	250	228	319	500	220	273	416
60	228	306	488	224	238	246	225	312	448	220	273	438
70	224	308	550	224	238	246	225	321	460	220	274	472
80	228	311	456	224	238	246	248	330	488	220	276	456
90	224	309	456	224	238	246	224	319	496	220	277	396
100	228	312	428	224	238	246	228	304	476	220	278	424

Table 7: Lists the best, average and worst fitness obtained in each of the various genetic algorithms. The double point crossover operator is used. Board is called P730-e5. The optimal solution obtained from exhaustive search is 220 seconds. All the values in the table are in seconds.

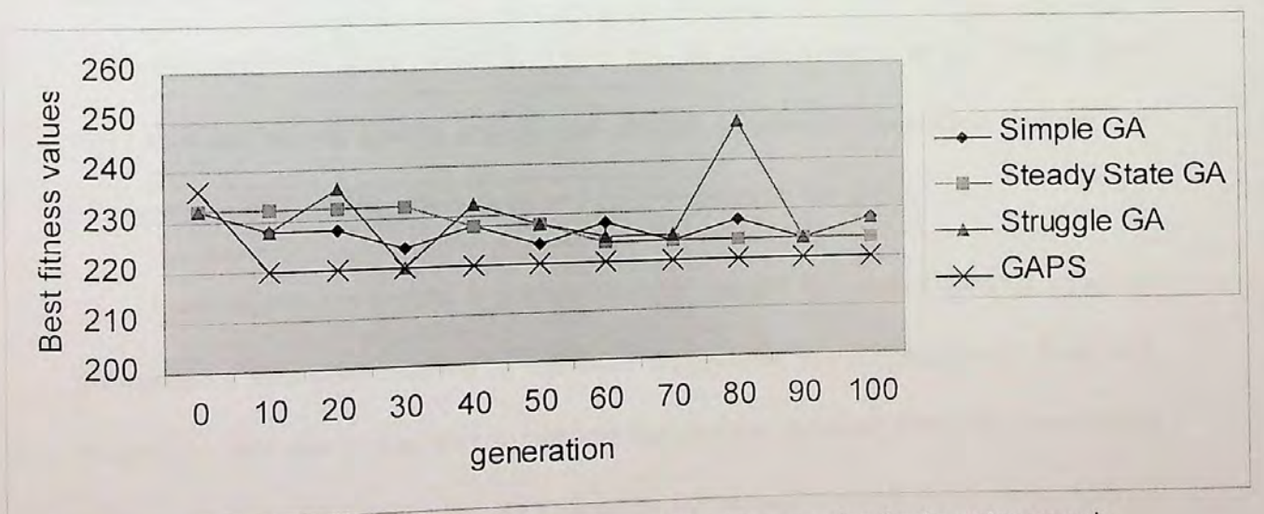

Figure 20: Compares the best fitness values obtained in each generation for the various genetic algorithms using the double point crossover operator on PCB P730-e5.

Gen.	Simple GA			Steady State GA			Struggle GA			GAPS		
	Best	Ave.	Worst	Best	Ave.	Worst	Best	Ave.	Worst	Best	Ave.	Worst
0	220	316	476	228	305	496	224	304	472	240	308	548
10	232	319	436	228	251	265	220	240	304	220	267	396
20	224	309	436	228	245	256	220	223	248	220	272	492
30	235	321	456	228	245	256	220	220	232	220	257	444
40	224	320	448	228	245	256	220	220	220	220	262	404
50	224	317	448	228	245	256	220	220	220	220	266	436
60	232	305	456	220	244	255	220	220	220	220	263	424
70	236	320	452	220	243	252	220	220	220	220	254	436
80	232	318	464	220	243	252	220	220	220	220	260	404
90	232	302	440	220	243	252	220	220	220	220	261	428
100	224	298	484	220	243	252	220	220	220	220	263	460

Table 8: Lists the best, average and worst fitness obtained in each of the various genetic algorithms. The generalized uniform crossover operator is used. Board is called P730-e5. The optimal solution obtained from exhaustive search is 220 seconds. All the values in the table are in seconds.

Table 8 lists the best, average and worst fitness obtained in each of the various genetic algorithms. The generalized uniform point crossover operator is used. PCB is called P730-e5. The optimal solution obtained from exhaustive search is 220 seconds. All the values in the table are in seconds. The performance of the Simple algorithm is almost similar to the previous results obtained when the single and double point crossover operators were selected. There was an improvement in the Steady State algorithm and it was able to achieve the optimal solution in generation number The performance of the Struggle method and GAPS were still the same.

From the above results it is clear that the Simple GA is not suitable cause its performance was poor relative to the other algorithms. As for the Steady State GA despite the fact that it was able to achieve the optimal solution using the generalized uniform crossover operator but its premature convergence in all the tests performed makes it easily trapped in a local optimal accordingly its results are not guaranteed.

This leaves us with the Struggle GA and GAPS methods. GAPS outperformed the Struggle GA when the single and double crossover operators were used.

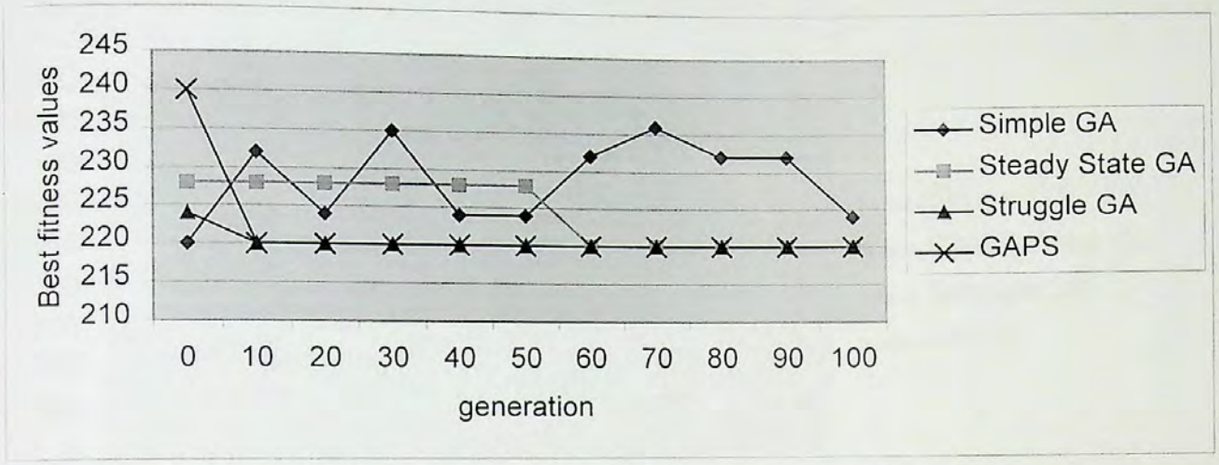

Figure 21: Compares the best fitness values obtained in each generation for the various genetic algorithms using the generalized crossover operator on PCB P730-e5.

Gen.	Simple GA			Steady State GA			Struggle GA			GAPS		
	Sing	Dou.	Uni.	Sing	Dou.	Uni.	Sing.	Dou.	Uni.	Sing.	Dou.	Uni.
0	0	0	0	0	0	0	0	0	0	0	0	0
10	1051	701	500	1472	1162	871	4196	3495	3855	1382	1322	1071
20	932	711	501	1282	1102	961	4827	3485	3976	1522	1282	1322
30	951	691	431	1412	2183	961	3845	3635	3906	1422	1112	1112
40	1102	741	440	1422	1853	901	4066	3635	3985	1472	1312	1052
50	972	731	461	1392	2033	1052	3826	3705	4076	1372	1272	1122
60	1002	731	440	1362	1783	1192	3886	3876	4256	1402	1252	1202
70	951	611	541	1852	1743	1282	3826	3535	3876	1472	1181	1112
80	971	711	430	1612	1922	1112	4306	3495	3966	1463	1282	1232
90	952	681	461	1562	1131	1152	3916	3575	3945	1412	1272	1061
100	861	671	490	1793	1472	1071	3805	3615	4386	1372	1282	1121

Table 9: Lists the runtime in ms taken by the Simple GA, Steady State GA, Struggle GA and GAPS.

As can be shown in table 9 the runtime needed by the Simple GA was the least. GAPS and the Steady State GA runtime are almost similar. The Struggle GA needed about triple the runtime taken by GAPS.

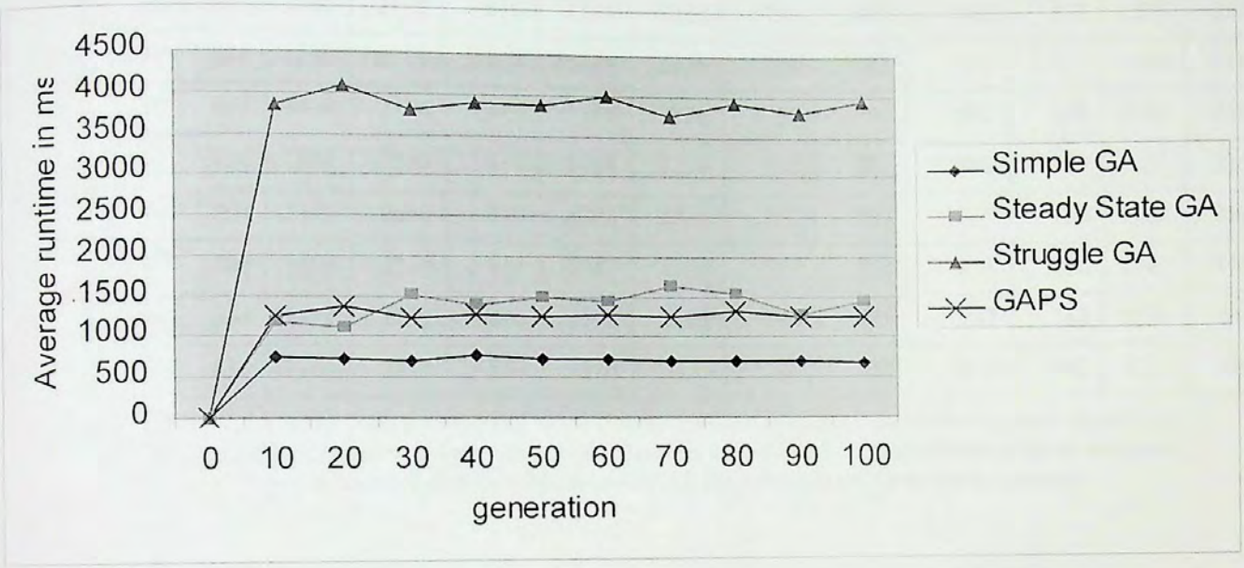

Figure 22: Compares the average runtime needed by each generation for the various genetic algorithms on PCB P730-e5.

Table 10 lists the best, average and worst fitness obtained in each of the various genetic algorithms. The single point crossover operator is used. PCB is called D665-2. The optimal solution obtained from exhaustive search is 742 seconds. All the values in the table are in seconds. The Simple GA still results in poor fitness values, the Steady State GA again got trapped in a local optima and the Steady State GA got the best results in all algorithms. GAPS result was also quite reasonable despite that it only achieved a near optimal solution.

Gen.	Simple GA			Steady State GA			Struggle GA			GAPS		
	Best	Ave.	Worst	Best	Ave.	Worst	Best	Ave.	Worst	Best	Ave.	Worst
0	826	1460	2303	810	1388	2226	882	1369	2261	872	1412	2212
10	844	1463	2233	810	1388	2226	824	980	1316	834	1307	2170
20	858	1374	2114	810	1388	2226	768	876	1197	792	1201	1932
30	889	1422	2198	810	1388	2226	756	831	1065	777	1275	2548
40	894	1390	2149	810	1388	2226	756	803	931	777	1176	2184
50	805	1428	2156	810	1388	2226	756	790	892	768	1165	2163
60	864	1394	2296	810	1388	2226	752	781	892	768	1239	2254
70	894	1375	2240	810	1388	2226	752	777	833	768	1204	2184
80	882	1366	2219	810	1388	2226	752	774	833	768	1161	2121
90	824	1442	2261	810	1388	2226	744	771	833	768	1270	2093
100	824	1361	2331	810	1388	2226	744	767	805	763	1215	2205

Table 10: Lists the best, average and worst fitness obtained in each of the various genetic algorithms. The single point crossover operator is used. Board is called D665-2. The optimal solution obtained from exhaustive search is 742 seconds. All the values in the table are in seconds.

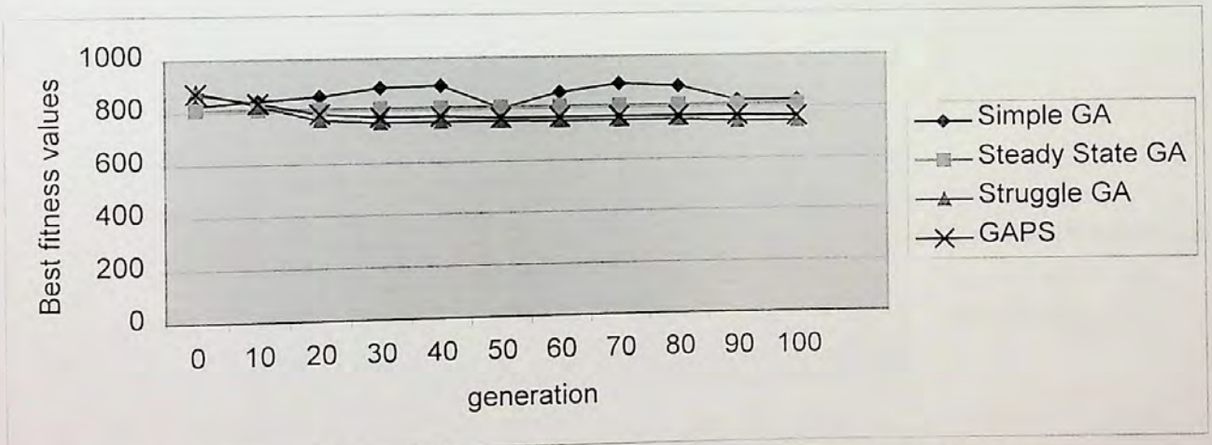

Figure 23: Compares the best fitness values obtained in each generation for the various genetic algorithms using the single point operator on PCB D665-2.

Table 11 lists the best, average and worst fitness obtained in each of the various genetic algorithms. The double point crossover operator is used. PCB is called D665-2. The optimal solution obtained from exhaustive search is 742 seconds. All the values in the table are in seconds. The Simple GA still results in poor fitness values, the Steady State GA again got trapped in a local optima and the Steady State GA got

the best results in all algorithms. GAPS results did not change much from the previous results when the single point crossover operator was used.

Gen.	Simple GA			Steady State GA			Struggle GA			GAPS		
	Best	Ave.	Worst	Best	Ave.	Worst	Best	Ave.	Worst	Best	Ave.	Worst
0	810	1388	2226	844	1400	2415	882	1369	2261	854	1417	2219
10	888	1427	2132	844	1400	2415	824	980	1316	812	1266	2240
20	876	1344	2289	844	1400	2415	768	876	1197	780	1203	2079
30	816	1398	2289	844	1400	2415	756	831	1065	772	1185	2121
40	840	1393	2289	844	1400	2415	756	803	931	772	1210	2219
50	900	1473	2177	844	1400	2415	756	790	892	772	1186	2009
60	780	1383	2275	844	1400	2415	752	781	892	772	1215	1956
70	848	1378	2345	844	1400	2415	752	777	833	770	1166	1897
80	900	1371	2219	844	1400	2415	752	774	833	770	1162	2022
90	836	1423	2254	844	1400	2415	744	771	833	768	1195	2212
100	822	1378	2233	844	1400	2415	744	767	805	768	1200	2149

Table 11: Lists the best, average and worst fitness obtained in each of the various genetic algorithms. The double point crossover operator is used. Board is called D665-2. The optimal solution obtained from exhaustive search is 742 seconds. All the values in the table are in seconds.

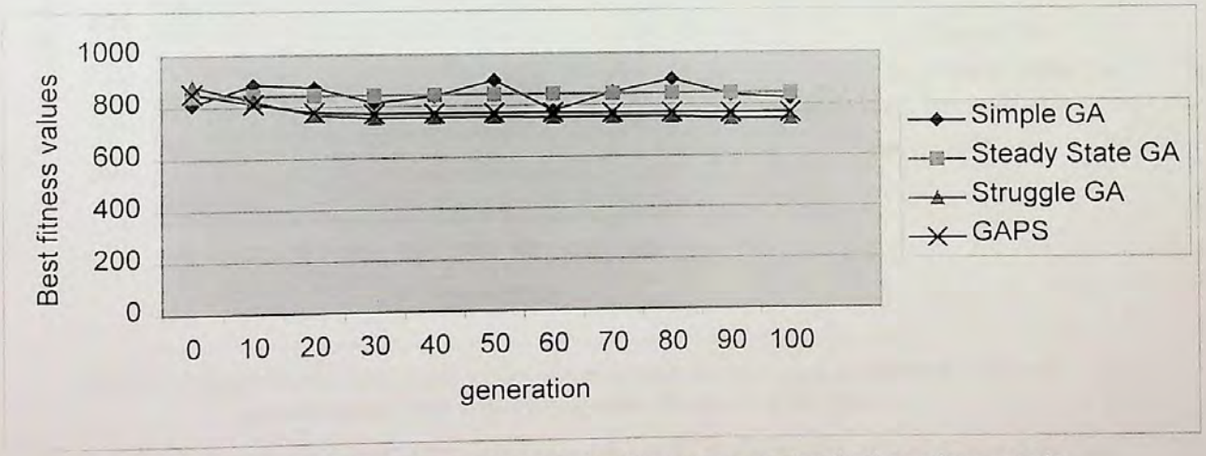

Figure 24: Compares the best fitness values obtained in each generation for the various genetic algorithms using the double point operator on PCB D665-2.

Table 12 lists the best, average and worst fitness obtained in each of the various genetic algorithms. The generalized uniform crossover operator is used. PCB is called D665-2. The optimal solution obtained from exhaustive search is 742 seconds. All the values in the table are in seconds. Using the generalized uniform crossover operator

resulted in much better results for the GAPS method. GAPS got almost the same results as the Steady State GA without the excessive need of large runtime.

Gen.	Simple GA			Steady State GA			Struggle GA			GAPS		
	Best	Ave.	Worst	Best	Ave.	Worst	Best	Ave.	Worst	Best	Ave.	Worst
0	804	1357	2191	798	1443	2240	882	1369	2261	834	1363	2247
10	912	1351	2114	798	1443	2240	824	980	1316	763	1110	2184
20	824	1355	2275	798	1443	2240	768	876	1197	756	1061	2156
30	888	1378	2052	798	1443	2240	756	831	1065	750	1064	2086
40	796	1359	2282	798	1443	2240	756	803	931	750	1023	1884
50	840	1490	2394	798	1443	2240	756	790	892	750	1072	2268
60	924	1427	2415	798	1443	2240	752	781	892	749	1096	2492
70	833	1410	2422	798	1443	2240	752	777	833	750	1037	2037
80	828	1396	2436	798	1443	2240	752	774	833	749	1100	2205
90	918	1427	2240	798	1443	2240	744	771	833	749	1029	2142
100	864	1426	2149	798	1443	2240	744	767	805	749	1041	2072

Table 12: Lists the best, average and worst fitness obtained in each of the various genetic algorithms. The generalized uniform crossover operator is used. Board is called D665-2. The optimal solution obtained from exhaustive search is 742 seconds. All the values in the table are in seconds.

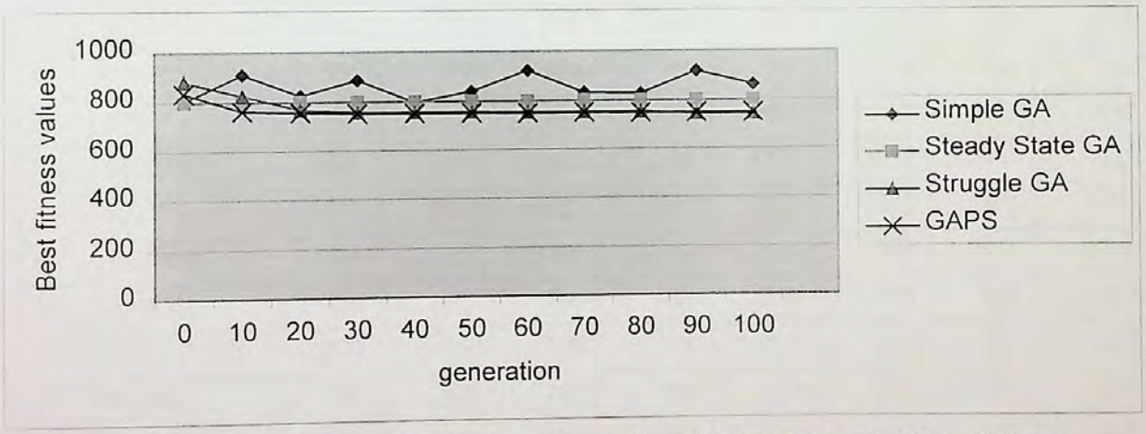

Figure 25: Compares the best fitness values obtained from the four genetic algorithms using the generalized uniform crossover operator. Board is called D665-2.

In addition to testing GAPS on various boards, the number of machines that can handle each part type was also varied. The number of machines that can handle each part type is an important factor in this problem type and it affects its runtime as will be shown in table 16. The varying of the number of machines in the production line does not have any effect on GAPS' performance or its runtime.

In case of 5 machines, 5 different machines in the production line can insert each part type. In case of 10 machines, 10 different machines in the production line can insert each part type. The production line was made up of 30 machines. The results shown in table 13 indicate that the increase of the number of machines that can handle each part type did not affect the GAPS' performance. GAPS was able to reach the optimal solution in almost all the cases. GAPS got near optimal solutions for boards D1123-W50 and P230A-02. Table 13 lists all the solutions obtained from GAPS using the generalized uniform crossover operator. Each column in the table gives the GAPS and the optimal solutions.

	Board Name	5 Machines		10 Machines		15 Machines		20 Machines	
		GAPS	Optimal	GAPS	Optimal	GAPS	Optimal	GAPS	Optimal
1.	Fmr5r4k	1814	1814	1707	1707	1707	1707	1707	1707
2.	VBPCBRWD	812	812	812	812	812	812	812	812
3.	EEPD	1574	1574	1574	1574	787	787	787	787
4.	d943_wg2	1314	1308	257	257	206	206	206	206
5.	8121346	951	951	658	658	658	658	658	658
6.	420285	950	950	950	950	950	950	950	950
7.	47847	978	974	481	487	487	487	487	487
8.	08204800	1802	1802	1802	1802	1433	1433	1433	1433
9.	837239	1522	1522	1522	1522	1522	1522	1522	1522
10.	COPR54	2718	2718	2740	2718	2718	2718	2718	2718
11.	AMU41	1456	1456	460	460	230	230	230	230
12.	D1123-W50	1092	1050	328	326	281	254	281	270
13.	XSP8U	777	726	732	726	732	726	726	726
14.	PSM01_C	1440	1440	203	167	167	167	167	167
15.	D665_2	749	742	242	241	203	203	203	203
16.	P732-D4.	268	268	72	72	72	72	75	72
17.	P730-E5	220	220	65	63	60	60	60	60
18.	P730-H8	196	196	68	61	68	61	63	61
19.	P230A-02	224	224	48	37	39	27	44	25

Table 13: Lists all the solutions obtained from GAPS using the generalized uniform crossover operator. Four different machine numbers were used in the tests.

Table 14 gives a list of all the solutions obtained from the Simple GA, Steady State GA, Struggle GA and GAPS. The runs were performed on the 20-machine case. The last column lists the optimal solutions. They all used the generalized uniform crossover operator. The Simple and the Steady State Genetic Algorithms were not able to reach the optimal solutions except in rare cases. For example the Simple GA was able to reach the optimal solution for board Fmr5r4k and the Steady State GA was able to obtain the optimal solution for board 420285. On the other hand the performance of the Struggle Genetic Algorithm was very close to that of GAPS. The Struggle Genetic Algorithm and the GAPS were able to reach the optimal solution in almost all the runs. In board P730-E5 GAPS was able to reach a better solution while in board P230A-02 the Struggle GA was able to reach a better solution.

	Board Name	Simple GA	Steady State GA	Struggle GA	GAPS	Optimal
1.	Fmr5r4k	1707	1710	1707	1707	1707
2.	VBPCBRWD	815	815	812	812	812
3.	EEPD	788	788	787	787	787
4.	d943_wg2	426	381	206	206	206
5.	8121346	660	671	658	658	658
6.	420285	952	950	950	950	950
7.	47847	495	488	487	487	487
8.	08204800	1448	1472	1433	1433	1433
9.	837239	1522	1522	1522	1522	1522
10.	COPR54	2724	2722	2718	2718	2718
11.	AMU41	255	248	230	230	230
12.	D1123-W50	445	376	281	281	270
13.	XSP8U	727	727	726	726	726
14.	PSM01_C	202	192	167	167	167
15.	D665_2	242	246	203	203	203
16.	P732-D4.	165	114	75	75	72
17.	P730-E5	111	110	87	60	60
18.	P730-H8	128	92	65	63	61
19.	P230A-02	77	66	38	44	25

Table 14: Lists the solutions obtained from the various genetic algorithms for the 20-machine case. They are all using the generalized uniform crossover operator.

As can be noticed from the results in table 14 the increase in the number of machines that can handle each part type did not have an effect on the runtime of the algorithm.

	Board Name	5 Machines	10 Machines	15 Machines	20 Machines	Average
1.	Fmr5r4k	1315	1348	1330	1626	1405
2.	VBPCBRWD	1519	1646	1578	1561	1576
3.	EEPD	1080	1151	1387	1090	1177
4.	d943_wg2	1188	1278	1255	1324	1261
5.	8121346	826	794	773	782	794
6.	420285	344	328	325	345	335
7.	47847	1325	1442	1372	1462	1400
8.	08204800	821	842	831	801	823
9.	837239	851	862	871	862	861
10.	COPR54	2504	2174	2443	2403	2381
11.	AMU41	1993	2293	1612	1662	1890
12.	D1123-W50	2563	2474	2734	2894	2666
13.	XSP8U	500	541	531	541	528
14.	PSM01_C	923	812	771	781	821
15.	D665_2	530	591	631	621	593
16.	P732-D4.	1061	1011	1252	1192	1129
17.	P730-E5	1211	1071	1152	1122	1139
18.	P730-H8	952	1422	1101	1022	1124
19.	P230A-02	691	741	731	712	718

Table 15: Lists the average runtime in ms for each generation in GAPS using the generalized uniform crossover operator. The last column gives the average runtime needed for the 4 different machine cases.

	Board Name	Number of Part Types	Average Runtime
1.	420285	35	335
2.	XSP8U	63	528
3.	D665_2	75	593
4.	PSM01_C	91	821
5.	P230A-02	92	718
6.	8121346	93	794
7.	08204800	94	823
8.	837239	98	861
9.	EEPD	123	1177
10.	P730-H8	127	1124
11.	P732-D4.	142	1129
12.	P730-E5	143	1139
13.	Fmr5r4k	145	1405
14.	47847	148	1400
15.	d943_wg2	150	1261
16.	VBPCBRWD	165	1576
17.	AMU41	172	1890
18.	COPR54	224	2381
19.	D1123-W50	266	2666

Table 16: Lists the GAPS average runtime for each board.

Table 16 gives a list of the average runtime needed for each board. As the number of part types increase the average runtime increases as well. The average runtime is almost 10 times the number of part types.

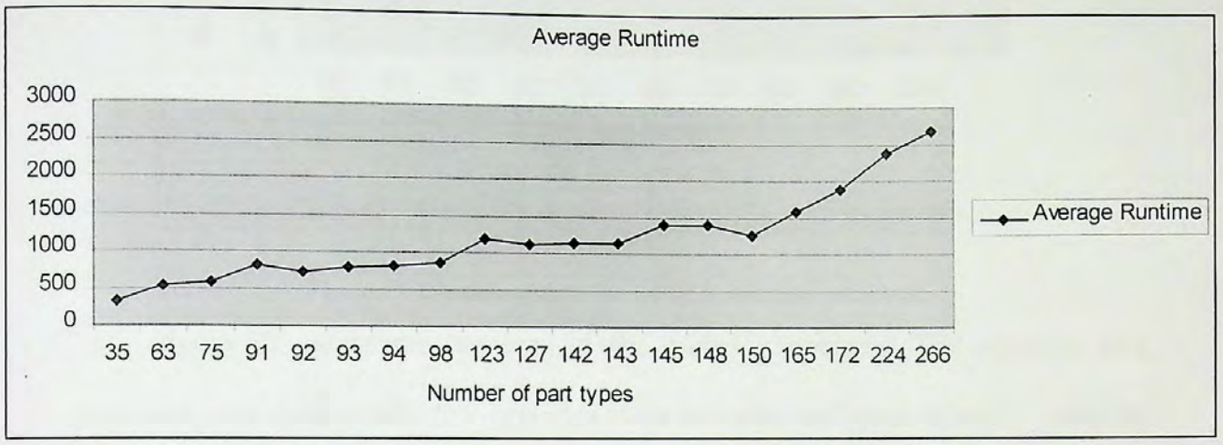

Figure 26: Shows GAPS average runtime using the generalized uniform crossover operator against the number of part types.

From all the above results and analysis it is clear that the GAPS outperforms all the other methods. The GAPS method obtained the optimal solution in about 90% of the runs and achieved a near optimal one better than that of the Simple GA and the Steady State GA in about 10 % of the runs. Since the runtime of the Struggle GA was by far more than GAPS and the results were very close so it can be stated that GAPS is the best solution method compared to the others.

GAPS Operators Performance

GAPS performed best (compared to exact solution methods) on the problems with large searching space and multiple optimum solutions. The extra combinations resulting from the large searching space as well as the multiple optimum solutions did not affect the genetic algorithm performance whereas it made the search more difficult for the branch and bound method.

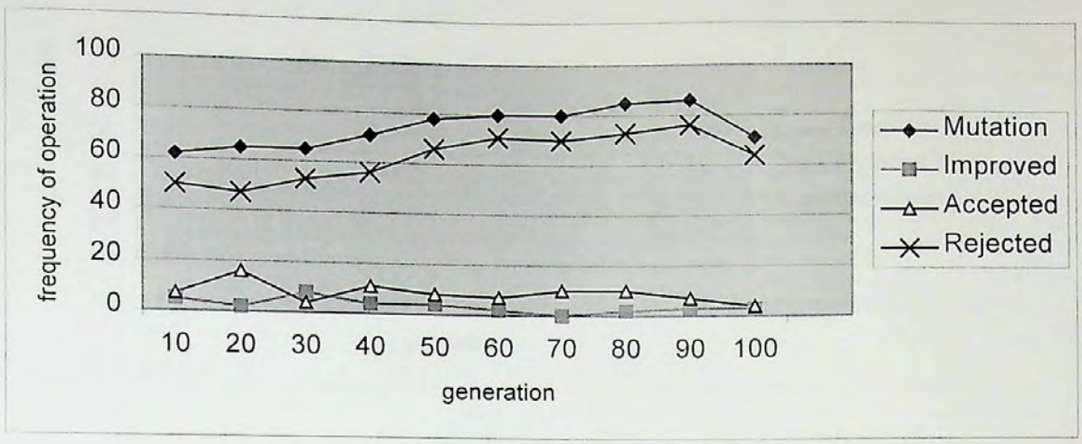

Figure 27: Classification of the mutation operator outcomes.

Figure 27 shows the outcome of the mutation operator. The rejection rate increases over time. Only few genomes were accepted and even a smaller number resulted in improvement.

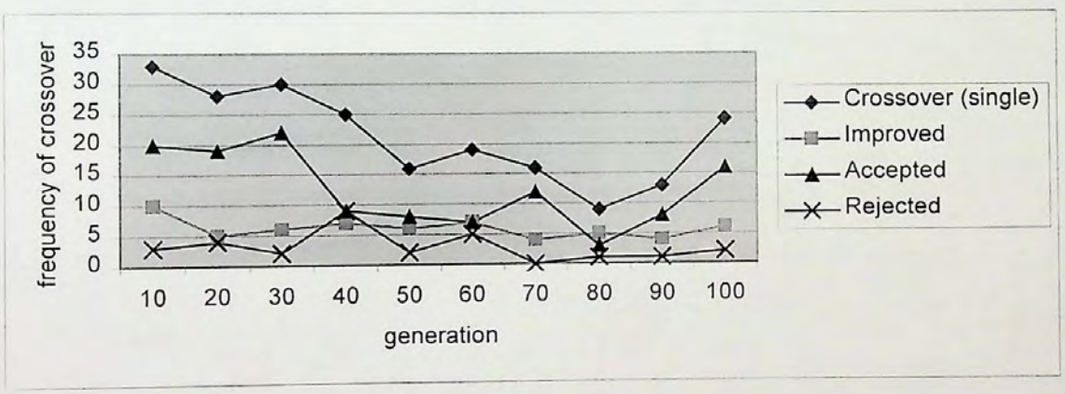

Figure 28: Classification of crossover outcomes.

Figure 28 shows the outcome of the generalized uniform crossover operators. The performance of the generalized uniform crossover operator resulted in a few rejected genomes. The acceptance criteria used was if the generated genome is different from the parent genome by less than 20% then it is accepted.

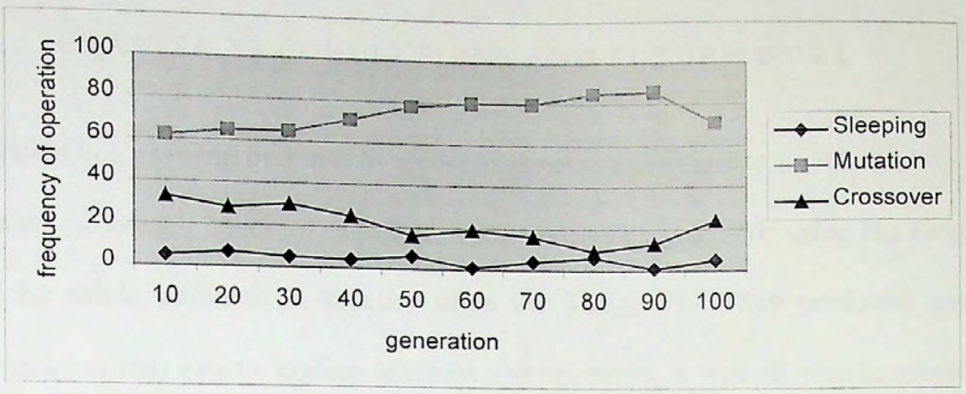

Figure 29: Relative frequency of operations over the generations.

Figure 29 shows the relative frequencies of reproduction operations performed in the GAPS. While the crossover frequency decreases, the mutation frequency increases. Sleeping occurs at an almost constant rate.

CHAPTER VI CONCLUSIONS AND FUTURE WORK

There is a growing tradition in applying genetic algorithms to a wide variety of scheduling problems. Much of the early work was concerned with using the GA to tackle the whole problem (in practice often the TSP). While this produced some useful insights into genetic coding schemes and operators, it was of little immediate practical value. Very often the main concern of the researchers involved was to gain a deeper understanding of the workings of genetic algorithms rather than to solve real problem. More recently genetic algorithms have been used in tandem with other methods to tackle realistic problems that were proving very difficult and needed new techniques. Results from this area of investigation are extremely encouraging.

Conclusion

This thesis started by highlighting the importance of planning and scheduling and that they are common to many different engineering domains. Then a description of the resource constrained scheduling problems in their general form was presented. It was important to study the general form problem before working closely with a special case of it which is the PCBs assembly line production scheduling. Reviewing the literature for previous related work associated with resource constrained scheduling problems proved out to be a good foundation for the analysis of the PCBs assembly line scheduling problem. Next, an overview about genetic algorithms was outlined.

PCBs assembly line production scheduling was presented in a separate chapter with details about its fundamentals and difficulties. In the literature review, the PCBs production scheduling problems were classified into four categories – single machine

optimization, setup strategy, component allocation to machines, and scheduling - and then a brief summary of the research done so far concerning the component allocation to machines category is presented. In addition, a summary of the commercial applications currently available for PCB assembly is presented from optimization point of view.

Since genetic algorithms performed rather well with a wide variety of scheduling problems it was motivating to apply it to the assembly line balancing problems. So I selected PCBs assembly lines as my case study. The main objective of this thesis was to provide a solution for the PCBs assembly line balancing problem using genetic algorithms. Accordingly a genetic algorithm called GAPS was developed for balancing production lines as elaborated in chapter V. The main components in this genetic algorithm are the genome representation as well as the approach of behavior driven interactions of the GA individuals in a structured population. There may exist an optimal mating scheme/population structure for a fixed population size and a problem under consideration. However, an optimal population structure appears to be highly problem dependent because the gene flow through the habitat strongly depends on the individuals ability to recombine themselves successfully.

After a comprehensive survey I found no related work in this area employed genetic algorithms accordingly the results of GAPS were compared against the exact solution obtained from a branch and bound method. A comparison was also presented between GAPS results and the results obtained from the Simple, Steady State and Struggle genetic algorithms. GAPS used three different crossover operators: single, double and uniform. The results using generalized uniform crossover operator were very encouraging.

As illustrated in chapter V GAPS performed best on the problems with large searching space and multiple optimum solutions. The extra combinations resulting from the large searching space as well as the multiple optimum solutions did not affect the genetic algorithm performance whereas it made the search more difficult for the branch and bound method. This suggests that the genetic algorithm is well suited to more-complicated problems.

The benefit of the GAPS solution method is twofold. It presents an efficient and robust optimization strategy, which can cope with varying constraints and varying problem sizes. Moreover, its extendable design provides a good starting point for further investigation in the remaining aspects that are not covered yet in the PCBs production-scheduling problem.

Future Work

What can be done to improve the genetic algorithm performance? Working on the representation in order to reduce the time taken by the crossover and mutation operators. Combining the genetic algorithm with another search algorithm may provide improvements.

In conclusion it is regarded that the genetic adaptation is a weak but robust optimization technique, which can meet the requirements of manufacturing systems. Genetic algorithms are capable to handle real world problems because the genetic representation of precedence relations among operations fits the needs of real world constraints in production scheduling. Moreover, genetic algorithms are applicable to a wide array of varying objectives and therefore they are open to many operational purposes. [Mattfield 1995]

The next step in the development of GAPS is placing it is a complete system that handles all the remaining categories in the problem that are not yet covered.

To conclude, here are six key topics for the future research on production planning in PCB assembly presented by [Smed 1999]:

Rolling horizon: although the production plan is made for a given period of time, the production rarely begins with an empty line, nor does the line remain empty, when the due date of the last job of the current plan expires. Yet, this rolling horizon framework is scarcely considered in the problem formulations of the PCB assembly literature.

Applicability: many of the solution procedures overlook the problems associated with the machine operation and the human workers. For example, partial setup strategy may, in some cases, provide the best theoretical solution for a given setup problem - but it may also result that the human operator, who is required to change few feeders whenever the board type changes, is prone to make more mistakes than if he performs larger feeder changeovers less often. Likewise, the technical considerations (e.g., machine code generation), in the main, are brushed aside in the literature, and thus the suggested solution procedures may have little applicability in actual production environments.

Dynamic production: reality rarely follows a plan: there are machine breakdowns, component shortages and maintenance delays, urgent prototype series surpass normal production, and the production plan itself can be subject to sudden alterations during the production period. Therefore, a practical production planning system must be able to cope with this kind of dynamic production and to give new or refined solutions whenever the integrity of the plan is challenged.

Multiple criteria: the bulk of research done in PCB assembly contemplates optimizing one - or rarely a few - criterion (e.g., component setup or due dates). In reality, there are usually several more or less important practical aspects, which affect the use of the solution. These aspects either defined the space of admissible solutions (e.g., release dates, operation duration, setup times and resource availability) or characterize the quality of scheduling decisions (e.g., due dates, productivity, frequency of tool changes and WIP levels). Some of these multiple criteria must be satisfied for a schedule to be valid, while others may not always be satisfied and might need to be relaxed.

User interaction: the importance of involving the human production planner in the decision making process earlier in this paper. To summarize, as long as the production planning systems are designed for not completely automated manufacturing processes (such as PCB assembly), the production planner must retain the final word on the production plan to be carried out. This means that the planner must be able to override the algorithmic solutions and effectively take the control if an exceptional situation requires it.

Integration: The lack of cooperation with other systems (such as CAD/CAM and inventory management) is a common reason why a new production planning system can be reluctantly accepted by the shop floor personnel. Production data should be interchanged automatically via a network – it must not depend on routinely done manual input. Here the key issue is the seamless integration of the production planning system to the other existing systems. [Smed 1999]

LITERATURE CITED

Ammons, J., M. Carlyle, L. Cranmer, G. DePuy, K. Ellis, L. F. McGinnis, C. A. Tovey, and H. Xu (1997): *Component allocation to balance workload in printed circuit card assembly systems*. IIE Transactions, 29(4): 265-75.

Askin, R., M. Dror, and A. J. Vakharia (1994): *Printed circuit board family grouping and component allocation for a multi-machine, open shop assembly cell*. Naval Research Logistics, 41:587-608.

Ben-Arieh, D. and M. Dror (1990): *Part assignment to electronic insertion machines: Two machine case*. International Journal of Production Research, 28(7): 1317-27.

Blazewicz, J. and J. K. Lenstra (1983): *Scheduling Subject to Resource Constraints: Classification and Complexity*. Discrete Applied Mathematics.

Bock, Stefan and Otto Rosenberg (1997): *A new distributed fault-tolerant algorithm for the simple assembly line balancing problem 1*. Paderborn University, Germany.

Brandeau, M and C. A. Billington (1991): *Design of manufacturing cells: Operation assignment in printed circuit board manufacturing*. Journal of Intelligent Manufacturing, 2:95-106.

Cleveland, G. A. and Smith, S. F. (1989): *Using Genetic Algorithms to Schedule Flow Shop Releases*. Proceedings of the International Conference on Genetic Algorithms.

Coffman, E.G, M. R. Garey and D.S. Johnsom (1997): *Approximation Algorithms for Bin Packing: A Survey*. Approximation Algorithms for NP-Hard Problems, D. Hochbaum (editor), PWS Publishing, Boston, 46-93

Davis, L. (1985): *Job-shop Scheduling with Genetic Algorithms*. International Conference on Genetic Algorithms and Their Applications, Pittsburgh, Lawrence Erlbaum Associates.

Demeulemeester, E. and W. Herroelen (1992): *A Branch-and-Bound Procedure for the Multiple Resource-Constrained Project Scheduling Problems*. Management Science 38(12): 1803.

Dorndorf, U. and Pesch, E. (1992): *Evolution Based Learning in a Job-shop Scheduling Environment*. Technical Report RM-92-019, Faculty of Economics, Limburg University.

Fox, B. and McMahon, M. (1991): *Genetic Operators for Sequencing Problems*. In G. Rawlins, editor, Foundations of Genetic Algorithms. Morgan Kaufmann.

Fox, M. S., and S. F. Smith (1984): *ISIS - a knowledge-based system for factory scheduling*. Expert Systems, 1(1): 25-49.

Goldberg, D. E. (1989): *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley.

Goldberg, D. E. and Lingle, R. (1985): *Alleles, loci and the Traveling Salesman Problem.* Proceedings of the International Conference on Genetic Algorithms, Hillsdale, N.J. Lawrence Erlbaum Associates.

Gorges-Schleuter, M. (1989*): ASPARAGOS: An Asynchronous Parallel Genetic Algorithm.* Proceedings of the International Conference on Genetic Algorithms.

Harvey, W. D., and M. L. Ginsberg (1995): *Limited Discrepancy Search. CIRL.* University of Oregon, Eugene, OR, USA.

Hayrinen, T., M. Johnsson, T. Johtela, J Smed and O. Nevalainen (1998): *Scheduling Algorithms for Computer-Aided Line Balancing in Printed Circuit Board Assembly.* University of Turku.

Held, M. and R. M. Karp (1962): *A Dynamic Programming Approach to Sequencing Problems.* Journal of the Society for Industrial and Applied Mathematics 10(1): 196-210.

Hillier, M. S. and M. L. Brandeau (1998): *Optimal component assignment and board grouping in printed circuit board manufacturing.* Operations Research 46(5): 675-89.

Hillis, W. D. (1985): *The Connection Machine.* MIT Press.

Holcomb, G. (1995): *Justifying flexible automation for PCB assembly.* Assembly Automation, 15(2): 14-6.

Holland, H. J. (1975): *Adaptation in Natural and Artificial systems.* The University of Michigan Press.

Huang, Chikong (1997): *A Study Of Line Balancing Problems Under Unlimited Number Of Workstation Using Simulated Annealing Procedure.* University of Science and Technology Yunlin, Taiwan.

Husbands, P., Mill F. G., and Warrington, S. W. (1987*): Knowledge Based Process-planning System.* Knowledge Based Expert Systems in Engineering: Planning and Design, pages 439-450. Computational Mechanics Publications.

Husbands, P. and Mill, F. (1994): *Scheduling with Genetic Algorithms.* School of Cognitive and Computing Sciences, University of Sussex, UK.

Johnsson, M. (1999): *Operational and Tactical Level Optimization in Printed Circuit Board Assembly.* Ph.D. thesis, University of Turku.

Johnsson, M., S. Peltonen, T. Leipala, and O. Nevalainen (1997): *Work load balancing of a generalized flexible flow line in printed circuit board production.* In R. V. Mayorga, editor, Proceedings of the Fifth IASTED International Conference on Robotics and Manufacturing, pages 382-9, Cancun, Mexico. IASTED, IASTED/ACTA Press.

Kim, Y-D., Lim H-G, Park M-W. (1996): *Search heuristics for a flow shop scheduling problem in a printed circuit board assembly process.* European Journal of Operational Research 91 pp. 124-143.

Klein, R. and A. Scholl (1996): *Maximizing the Production Rate in Simple Assembly Line Balancing -a Branch and Bound Procedure*. European Journal of Operational Research 91 pp. 367-385.

Klein, R and A. Scholl (1998): *PROGRESS Optimally solving the Generalized Resource Constrained Project Scheduling Problem*. Darmstadt University

Kolisch, R. (1995): *Project Scheduling under Resource Constraints*. Heidelberg, Physica-Verlag.

Lawler, E. L. and D. E. Wood (1966): *Branch and Bound Methods: A Survey*. Operations Research 14(4): 699-719.

Ling (1991): *Constructing a College Timetable by Integrating Two Approaches: Logic Programming and Genetic Algorithms*. Technical Report M.Sc. Thesis, School of Cognitive and Computing Sciences, University of Sussex.

Lofgren, C. and L. F. McGinnis (1986): *Soft configuration in automated insertion*. In Proceedings of the 1986 IEEE International Conference on Robotics and Automation, pages 138-42, San Francisco, CA.

Mansour, N. and Fox, G. (1991*): A Hybrid Genetic Algorithm for Task Allocation in Multicomputers*. Proceedings of the International Conference on Genetic Algorithms.

Mattfeld, Dirk C. (1995): *Evolutionary Search and the Job Shop – Investigations on Genetic Algorithms for Production Scheduling*. Physica-Verlag.

McGinnis, F., J. C. Ammons, M. Carlyle, L. Cranmer, G. W. DePuy, K. P. Ellis, C. A. Tovey, and H. Xu (1992): *Automated process planning for printed circuit card assembly*. IIE Transactions, 24(4): 18-30.

Muller-Merbach, H. (1967): *Ein Verfahren zur Planung des optimalen Betriebsmitteleinsatzes bei der Terminierung von Gro§projeckten Zeitschrift fur wirtschaftliche Fertigung*, Vol. 62, pp. 83-88, 135-140.

Nakano, R. and T. Yamada (1991): *Conventional Genetic Algorithms for Job-shop Problems*. Proceedings of the International Conference on Genetic Algorithms.

Neumann, K. (1990): *Stochastic Project Networks - Temporal Analysis, Scheduling, and Cost Minimization*. Berlin, Springer-Verlag.

Patterson, J. H. (1984): *A Comparison of Exact Approaches for Solving the Multiple Constrained Resource*. Project Scheduling Problem. Management Science 30 (7): 854.

Rechenberg, I. (1973): *Evolutionsstrategie: Optimierrung Technischer Systeme Nach Prinzipien der Biologischen Evolution*. Friederich Frommann Verlag, Stuttgart.

Reeves, C. (1992): *A Genetic Algorithm Approach to Stochastic Flow-shop Sequencing*. In Colloquium on Genetic Algorithms for Control and Systems Engineering, London.

Rosmaita J. Grefenstette, R. Gopal and D. Van Gucht (1985): *Genetic Algorithms for the Travelling Salesman Problem.* Proceedings of the International Conference on Genetic Algorithms, Hillsdale, N.J., Lawrence Erlbaum Associates.

Scheer, A. (1989): *Enterprise-Wide Data Modeling: Information Systems in Industry.* Springer Verlag, Berlin Heidelberg.

Shmoys, D.B., Stein C., Wein J. (1994): *Improved Approximation Algorithms for Shop Scheduling Problems.* Siam Journal on Computing, 23, 617-632.

Slowinski, R. and J. Weglarz, Eds. (1989): *Advances in Project Scheduling.* Amsterdam, Elsevier.

Smed, J., M. Johnsson, T. Johtela and O. Nevalainen (1999): *Techniques and Applications of Production Planning in Electronics Manufacturing Systems.* University of Turku.

Smith, D. (1985): *Bin Packing with Adaptive Search.* Proceedings of the International Conference on Genetic Algorithms, pages 202-207. Lawrence Erlbaum.

Sprecher, A. (1994): *Resource-Constrained Project Scheduling: Exact Methods for the Multi-mode case.* Lecture Notes in Economics and Mathematical Systems 409.

Sprecher, A. and A. Drexl (1996): *Manuskripte aus den Instituten fur Betriebswirtschaftslehre der UniversitŠt Kiel.* Kiel, UniversitŠt Kiel.

Stinson, J. P., E. W. Davis, et al. (1978): *Multiple Resource-Constrained Scheduling Using Branch and Bound.* AIIE Transactions 10 (3): 252-259.

Syswerda, G. (1990): *Schedule Optimization Using Genetic Algorithms.* The Handbook of Genetic Algorithms, Van Nostram Reinhold.

Tavares, L. V. and J. Weglarz (1990): *Project Management and Scheduling: A Permanent Challenge for OR.* European Journal of Operational Research.

Teng, G. and S. S. Garimella (1998): *Manufacturing cost modeling in printed wiring board assembly.* Journal of Manufacturing Systems, 17(2): 87-96.

Wall, Matthew B. (1996): *A Genetic Algorithm for Resource-Constrained Scheduling.* Doctoral Thesis, Massachusetts Institute of Technology, Cambridge, MA, USA.

Watkins, R and J. K. Cochran (1995): *A line balancing heuristic case study for existing automated surface mount assembly line setups.* Computers & Industrial Engineering, 29(1-4): 681-5.

Whitley, D. and T. Starkweather (1990): *Genitor II: A Distributed Genetic Algorithm.* Journal of Experimental and Theoretical AI, 2:189-214.

Wittrock, R. (1988): *An adaptable scheduling algorithm for flexible flow lines.* Operations Research, 36(3): 445-53.

Wren A. and D. Wren (1990): *Genetics, Structures and Covers -- an Application to Scheduling*. Technical Report 90.23, School of Computer Science, University of Leeds.

APPENDIX A - GLOSSARY

allele

One of a set of possible values for a gene. In a binary string genome, the alleles are 0 and 1.

chromosome

A set of information that encodes some of an individual's traits. In evolutionary algorithms, chromosome is often used to refer to a genome.

crossover

A genetic operator that generates new individuals based upon combination and possibly permutation of the genetic material of ancestors. Typically used to create one or two offspring from two parents (sexual crossover) or a single child from a single parent (asexual crossover).

crowding

A niching method in which speciation is encouraged by replacing individuals in the current population with newly generated individuals that share the same characteristics.

deme

A population of individuals. Members of a deme typically share common traits.

earliest finish

The earliest time at which an activity can be completed.

earliness

The amount of time between the due date and actual finish time for an activity.

evolutionary algorithm

A class of stochastic algorithms based on simplifications of natural evolutionary processes such as selection, survival-of-the-fittest, mating, mutation, and extinction.

exploitation

Local search.

exploration

Global search.

feeder

A feeder is a container that is mounted on a production machine and contains the parts to be placed on the PCB being assembled.

gene

The smallest unit in a genome. In a binary string genome, the bits are genes.

In an array of characters, each character in the array is a gene.

genetic algorithm

An evolutionary algorithm in which a population of individuals is evolved using selection, crossover, and mutation. Originally devised as a model of evolutionary principles found in Nature, genetic algorithms have evolved into a stochastic, heuristic search method. A genetic algorithm may operate on any data type with operators specific to the data type.

genetic programming

The use of genetic algorithms to evolve programs. Genetic programming typically uses tree genomes (or tree genomes in combination with other data structures) to represent parse trees. A genetic algorithm then evolves trees using the parsed tree's performance as the objective function.

genome

A complete representation of the information required to characterize the traits of an individual. In evolutionary algorithms, a single solution to a problem.

genotype

The genetic traits of an individual. In a binary-to-decimal genome, the bits are the genotype.

idle time

The amount of time a resource is not actually working on an activity.

job

A job stands for one product to be produced with a defined quantity within a defined time.

latest finish

The latest time at which an activity can be completed.