

PART 1

Introduction

Chapter 1 presents the conventional optimization techniques for solving combinatorial problems, and their limitations. Adaptive memory programming approaches that address these shortfalls are then presented. The fundamentals of adaptive memory programming are fully discussed. These are generally accepted as the yardstick for determining whether or not a method qualifies as being intelligent. Neural-networks, genetic algorithms, tabu search, and ant systems are presented as being adaptive memory programming. These are briefly discussed. Hybrid systems, which are extension of adaptive memory programming are presented. A full treatment of production planning and control is given so that readers can understand the underlying theory of the areas of application of the book. The material on production planning and control is sufficient for two semester modules at the undergraduate final year level. This chapter therefore, sets the scene for what follows in the remaining parts of the book.

This page is intentionally left blank

Chapter 1

Introduction to Adaptive Memory Programming and Production Planning and Control

1.1. Production Planning Control within Integrated Manufacturing Framework

Computer Integrated Manufacturing (CIM) requires integration among the basic manufacturing functions (technical and operational tasks of an industrial

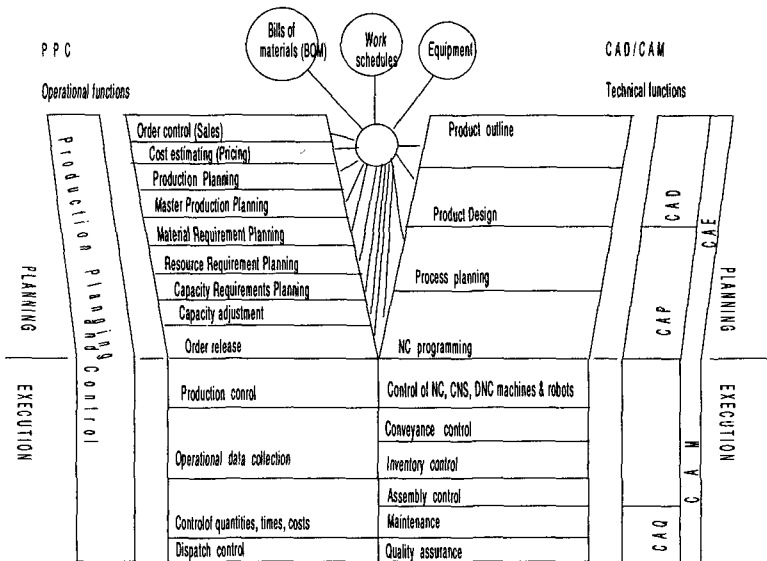


Fig. 1.1. Basic integrated manufacturing functions.

enterprise) shown in Fig. 1.1. Production planning and control system (PPC) refers to the operational tasks represented in the left part of the Y. The more technical activities are characterized by the various computer aided (CAX)-concepts in the right fork of the Y. The PPC system is normally determined by order handling, whereas the CAX-components support product description and production resources.

As can be seen from the integrated manufacturing framework in Fig. 1.1, production planning and control encompasses important aspects of an industrial enterprise. Production planning and control is the hub that ties together product design and manufacturing engineering. It is concerned with translating business plan into reality. The planning horizon involved is shorter, usually in terms of the next three to six months in the near future. The functions of production planning and control (PPC) system are shown in Fig. 1.2. In order to understand the whole process involved in production planning and

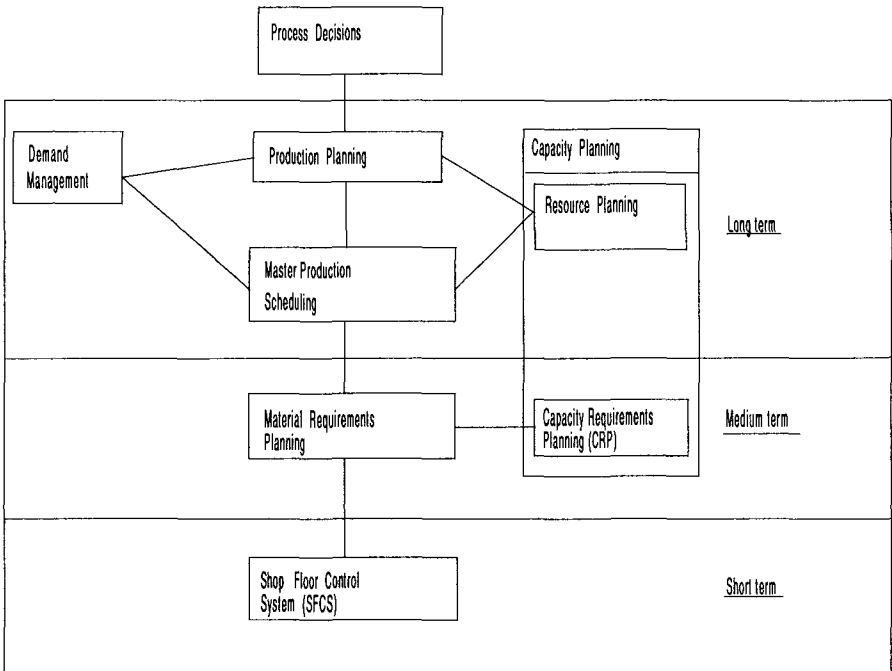


Fig. 1.2. Production planning and control framework.

control, the different functions are summarized in this chapter, but are given full treatment in Chapters 2 and 3.

1.1.1. Demand management

Forecasting is the prediction of the future by considering past data. It is an input to production planning, hence accurate methods of forecasting have to be adopted by management in order to achieve a good plan. Forecasting has impact on planning for capacity, schedule and procurement of raw materials. There are basically two groups of forecasting methods. The first, is the time-series in which history is considered to repeat itself, so that the future is considered to be some kind of continuation of the past. The second, is the explanatory type, in which the future is predicted by simply understanding the factors that give explanation for variation in some variables of interest. Simple moving average, single exponential smoothing, and seasonal exponential smoothing techniques fall under the first group, while simple regression falls under the second group. These forecasting techniques are discussed in Chapter 2 as important tools for business planning in the remaining part of this section.

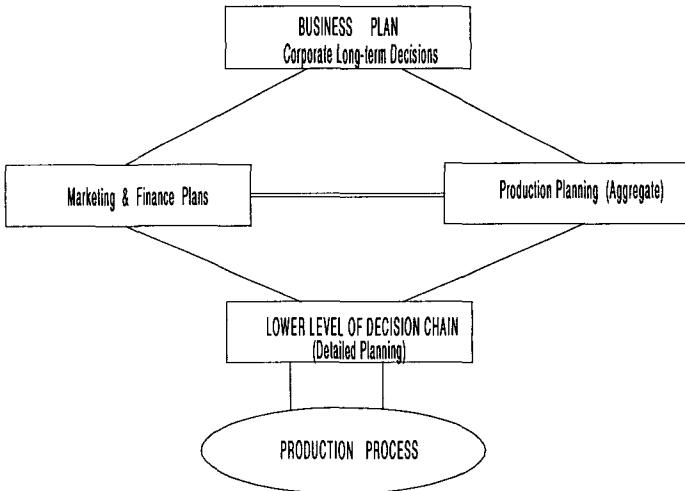


Fig. 1.3. Simplified decision-making process.

1.1.2. *Production planning*

Within the hierarchical decision process in the production planning and control framework, one of the first actions of planning is the elaboration of an aggregated production plan. Figure 1.3 summarizes some of the main modules that constitute a typical strategic level within a hierarchical framework (Filho, 1999). The production plan module interacts strongly with the upper and lower levels of the framework. The production plan aims at achieving the goals set by the business plan. This production plan is used as a target to be followed by the lower level. This target becomes the constraint to be satisfied during development of the detailed planning.

The production plan is conceived over a long-term horizon, between one to two years with monthly or quarterly discrete time buckets. Under this time horizon, enormous quantity of data is available to state the production plan, but a great part of it is not exactly known. Hence, the planning is done using both aggregate and forecasting raw data from marketing department.

The main objective of the aggregate production plan is to tune the resources available in the industrial enterprise so as to meet the fluctuating market demand requirements and minimize the expected total production costs. The advantages realized from aggregated forecast demand for production planning are as follows (Gelders and van Wassonhove, 1981):

- improvement of the plant manager's overall insights
- minimizes the plant manager's overall computational efforts
- reduced variance.

1.1.3. *Master production scheduling*

While the production plan was considered as being clustered, with regard to material, capacity requirements, and product groups, the master production schedule (MPS) is de-clustered. The MPS is a specific statement of the exact individual end items that are to be manufactured in each time period. These time periods are now much shorter than the production planning time horizon and are usually in terms of weeks. The summation of the de-clustered items must sum up to the same clustered end result items.

1.1.4. *Final assembly scheduling*

When customer requirements for finished products can be predicted with some degree of certainty, it will be more beneficial to plan the final assembly schedule (FAS) of subassemblies or groups of options. FAS has the same features as MPS except that the planning horizon is usually as short as a week ahead of time, with exact plan for the option combinations for each finished end item to be produced. In just-in-time (JIT) operations, the time horizon for FAS reduces to as short as hours within each working day of the week.

1.1.5. *Material requirement planning*

The MPS is input to material requirement planning (MRP). In order to execute the MPS, it is necessary to plan to have the material required, parts, and assemblies required for production of the finished products. Decisions have to be reached on which parts or assemblies are to be bought from outside the company and which ones are to be manufactured in-house. Orders have to be placed in good time for these parts, assemblies, or materials so that the MPS can be accomplished. Parts, materials, or assemblies that are to be supplied by outside vendors have to be ordered in good time by the purchasing department. Similar items that are to be manufactured in-house have to be scheduled into production and made on various machines in good time in order to fit into the planning horizon of the final assembly schedule. In MRP, the basic question to answer, is, when do we place order for materials, parts or assemblies so that they can be received for production within the lead-time in order to meet customer requirement?

1.1.6. *Resource requirement planning and allocation*

Resource requirement planning is a typical long-term activity, which provides the capacity required for manufacturing products and meeting market demand. Using a production plan as an input, resource requirement planning carries out an evaluation of aggregate resources (such as gross labour-hours, capital expansion, machines, warehouse space, etc.) in preparation for future production.

1.1.7. *Rough-cut-capacity planning*

Strictly related to resource requirement planning is rough-cut-capacity planning which, even though on a shorter term, provides information on how resource levels should be changed to ensure the execution of the master production schedule (Vollmann *et al.*, 1991). The concept of rough cut capacity planning (RCCP) used in production planning applies to operational planning. The essence of RCCP is to check that the MPS takes into consideration capacity requirements. However, since MPS planning horizon is in terms of weeks it may not be possible to accurately estimate the capacity of individual machines until a further detailed level of planning is done.

1.1.8. *Capacity requirement planning*

When material requirement planning is completed, it would be necessary to check if there are available capacity to meet both the MPS and MRP. Capacity requirement planning accomplishes this by estimating total capacity needed by the individual machine or work centre and equating it to the capacity required to balance the MPS and MRP. Unloaded machines and work centres are identified and it may be necessary at this stage of planning to move jobs from an over-loaded machine or work centre to under-loaded machine or work centre. The planning at this stage is very dynamic and usually puts much pressure on operations managers, because last minute decisions must be taken to meet production requirements. It may be necessary to schedule overtime or hire temporary workers. When the MPS is simply non-doable, it may be altogether revised in the interest of the company.

1.1.9. *Production control*

Close to the shop floor operations is the production scheduling which is the actual planning of the sequence of production of the various parts on machines or work centres. Production scheduling involves detailed determination of the sequence in which finished products will be produced, the employee work times and size, and the individual machine operation times. Mistake made at this time costs the company much money, and hence this aspect of planning is given much attention by management. This is where the strategic plan, which has cascaded through the business plan and production plan, down to the operational plan is executed.

For a more extensive coverage of production planning and control, interested readers may consult Chapters 2 and 3, Melnyk and Denzler (1996), Vonderembse and White (1988) and Wild (1992).

1.1.10. *New developments*

We have presented an overview of production planning and control as it takes place in the industrial enterprise that has the goal of delivering finished product or service to a satisfied customer. It is projected that in the near future, an engineer will perform a multitude of highly specialized tasks, with each of these having more than one disciplinary flavor. The search for building systems that can function automatically has attracted a lot of attention over the centuries and created continuous research activity. The current trend is to build autonomous systems that can adapt to changes in their environment. Although there is much to be done before we can reach this point, the desire to achieve fully automated manufacturing system is here to stay.

Computer Integrated Manufacturing (CIM) systems of the 21st century will demand more flexibility in product design, process planning, process planning and control, manufacturing operations and process control. This may be achieved through integrated software and hardware architectures that generate current decisions based on information collected from the manufacturing systems environment and then execute these decisions by converting them into signals transferred through a communication network.

With regard to CIM, the realization of this integration principle necessitates the creation of the relevant data links between the technical areas of product design, process planning and production and the corresponding operational processes, such as production planning and control. This means that information systems which are in themselves already partly integrated must now be unified with each other, due to the fact that technical and operational activities involve increasing interaction within the processing chain of a given customer order. Integrated manufacturing has not yet reached this state. However, the urge for achieving this strategic vision is transferred into the term “intelligent systems” that we are now using in the late 1990s and early 2000s.

Knowledge-based systems, our first effort in this endeavour, were not sufficient in the 1980s to generate the “intelligence” required, so our quest still continues. The approach to improve our quest for intelligent systems is to merge the old with the new technologies. Adaptive memory programming

are other technologies that need to be integrated to improve our quest for intelligent systems.

Newly emerging-technologies, such as adaptive memory programming, will be some of the contributions in the quest of achieving fully adaptive or intelligent manufacturing systems in the 21st century. This book, *Emerging Optimization Techniques in Production Planning and Control*, presents in a unique way, *adaptive memory programming* approaches and their impact in the design of autonomous manufacturing units to realize our quest of building adaptive systems.

1.2. Conventional Combinatorial Optimization Techniques

The theory of optimization encompasses the quantitative study of optima and methods for finding them. Therefore, when we say we want to optimize a function or process, we are in effect seeking to improve performance toward some optimal point or points. Two issues are important in optimization: the optimal point which is the goal, and seeking improvement on how to get there. Over the past two centuries, theorists interested in optimization have emphasized convergence, that is, reaching the optimum. They paid less attention to *provisory improvements* that drive the process of optimization towards the destination or optimum.

Many real life optimization problems encountered in practical settings such as transportation, logistics, production and task scheduling, telecommunications and financial planning have large search spaces. For any search problem, the search space is the set of possible solutions, while the solution space is the subset of the search space that contains actual solutions. The underlying problem in real life optimization problems such as in the examples cited above, is that it is impossible to sample every point in the search space. If all the points in a search space are examined, it is possible to find an optimum solution. But as this is impossible for this class of problems, we must reject techniques that emphasize convergence. This leads to the problem of selecting points to search, which give a “good enough” performance in a reasonable time. In difficult optimization problems encountered in practical settings, convergence to the optimum is not an issue. What is important is getting a *good enough* solution, within a reasonable time and so improve the expectation of overall return in the long run.

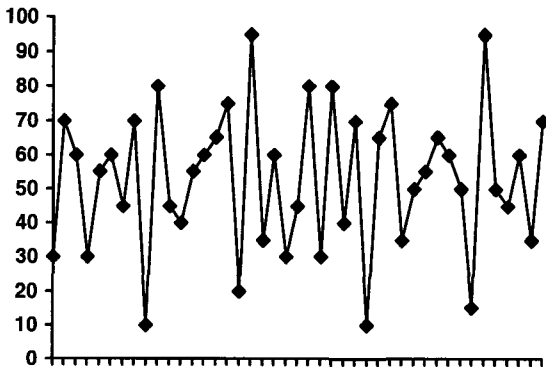


Fig. 1.4. Noisy and discontinuous functions that confound traditional methods.

We have only looked at one aspect of difficult real life optimization problem, which is the large search space that is impossible to search on a point-to-point basis for the optimum solution. The other aspect of this class of problems is that they abound with discontinuities and vast multi-modal, noisy search spaces as shown in Fig. 1.4. These search spaces are clearly described by less friendly calculus functions. Here, we are faced with functions of *discrete* variables. Combinatorial optimization is the search for optima of functions of *discrete* variables. From now onward, we take it for granted that the problems we are solving are: (1) combinatorial optimization problems (2) of infinite search spaces that cannot be exhaustively searched by a finite machine such as the computer. We conclude that we are faced with a dilemma of putting aside all traditional techniques of seeking for an optimum solution and seeking methods that will search only subset of the search space in order to reach a near-optimum that is *good enough* within a reasonable time.

The remaining parts of this chapter discuss conventional combinatorial optimization techniques, intelligent optimization fundamentals, adaptive memory programming, and combinations of adaptive memory programming approaches that result in hybrid systems.

We have concluded that we are left in a situation, where in practical settings many larger-scale combinatorial optimization problems cannot be solved to optimality, due to the constraints that prohibitive amounts of computation times are required to search for the optimum solution. For example, finding the best sequence for producing n jobs on a machine for a given performance measure

(e.g. completion time) requires $n!$ combinations. Suppose there are 100 jobs, how many combinations are required? The answer is simple: there are $100!$ (factorial) combinations that have to be considered. We begin to see the difficulty faced by the computer power if an attempt is made to search each point in the search space.

There are two conventional combinatorial optimization techniques: optimization algorithms and approximation algorithms. These are now discussed in some details in order to have understanding in issues involved in solving this class of problems.

1.2.1. *Optimization algorithms*

Optimization algorithms are capable of solving a combinatorial optimization problem, yielding a globally optimal solution in a possible prohibitive amount of computation time. Two well-known optimization algorithms are *branch-and-bound* and *A* algorithm*. Both of these schemes use tree structures made up of nodes connected by arcs. In the *A** algorithm, all the existing paths that have been found so far are evaluated and the shortest one is chosen and advanced. In this way some branches of the tree are never explored, hence the search time is reduced. Moreover since many alternative paths are explored and are never entirely dropped, optimal solution is guaranteed. The *A** algorithm is therefore, a very powerful breath-first tree-search technique when evaluation is available at each node of the tree structure. The *branch-and-bound* technique commences with a partial solution at the root node. As branches progress into more complete partial solutions, new terminal nodes replace old ones. Although these optimization algorithms guarantee optimum solutions, they are less attractive in solving many large-scale combinatorial optimization problems. This is because, this class of problems are solvable by an amount of computational effort that is bounded by a polynomial function of the size of the problem.

1.2.2. *Approximation algorithms or heuristics*

When designing a heuristic for a combinatorial optimisation problem one can either build a method that is specific to the problem under study or adapt some general existing schemes. In the latter case constructive, sequential and adaptive techniques are search strategies that have successfully been used for the solution of difficult problems.

Constructive methods build feasible solutions by starting from an “empty” solution and by inserting repeatedly a new component into the current partial solution. Sequential methods start from an initial feasible solution and move step by step from one solution to a neighbour solution by performing some elementary modifications.

Approximation algorithms can be divided into two categories: *constructive* and *neighbourhood* or *local search*. These two categories are now discussed.

Neighbourhood or *local search* techniques have an operation called a *move*. The move operation defines a neighbourhood N_i for each configuration i in one iteration. Commencing from a given configuration, a possible transition from the current configuration to a configuration chosen from the neighbourhood of the current configuration is examined. If the objective function evaluated for this neighboring configuration is less than the objective function evaluated for the current configuration, then this neighbour replaces the current configuration, otherwise another neighbour is selected and compared for its evaluated objective function. The algorithm continues until it finds a configuration whose evaluated objective function is better than its neighbours. Such a neighbourhood search that we have just described is the *steepest descent*. Although the steepest-descent approach has attractive features it is impracticable in certain practical settings because it is computationally too expensive, as where the neighbourhood contains many elements or each element is expensive to evaluate. Local search method also suffers from the disadvantage of terminating in a local optimum for which we may not know how far it is from the global optimum solution. Neighbourhood describes all moves that can be made from a current solution. Suppose in 6-job flow shop problem, the current solution (i.e. sequence) is $\{2, 3, 1, 6, 5, 4\}$. If an insert-move is adopted, then removing a job from the i th position and inserting it in the j th position describes a neighbour. For example, removing a job from the fifth position and inserting it in the second position results in a sequence $\{2, 5, 3, 1, 6, 4\}$. The *move* described illustrates the concept of neighbourhood as follows:

<i>Neighbourhood 1</i>	1	2	3	1	6	5	4
<i>Neighbourhood 2</i>	2	2	5	3	1	6	4

The first neighbourhood is made up of sequence $\{2, 3, 1, 6, 5, 4\}$. By inserting the fifth job in the second position, a new neighbourhood is obtained

which is made up of sequence $\{2, 5, 3, 1, 6, 4\}$. We defer detailed discussions on neighbourhood or local search until we get to later chapters.

Constructive methods build feasible solutions from an initial partial solution by inserting repeatedly a new component into current partial solution. Most constructive methods can be conceived to be descent methods, where each constructive step toward a more complete solution therefore is an improving step in the neighbourhood being considered. The final constructive step becomes a local optimum since no better solution is obtained by going beyond a complete construction. The concept of constructive method is best understood using the curtailed-enumeration algorithm of NEH (Nawaz, Enscofe, and Ham, 1983) for scheduling n jobs on m machines. We defer detailed discussions on curtailed-enumeration algorithm of NEH until we get to later chapters.

Both constructive and neighborhood or local search techniques give “good solutions” in short computation times, for combinatorial optimization problems. But we recall that the main goal of optimization is to seek for interim improvements that drive the process towards the destination, the optimum solution. This underlying philosophy is the motivation for research in seeking for adaptive memory programming in the past few decades. This is the very essence of this book: to present modern or emerging-intelligent techniques that are preferred to both constructive and neighborhood or local search techniques for solving practical real life combinatorial optimization problem. That is, we emphasis on improvements on provisory solutions with the goal of moving nearer to the optimum solution in a short time. Constructive and neighborhood or local search methods may obtain “reasonable” solution in very short time but such solution may be local optima for which we are not sure how far they are from the global optimum. We need some mechanisms that can facilitate the “jumping out” from local optimum with the hope that our solution moves toward the global optimum. Not all techniques that can provide this “jumping out” mechanisms qualify for adaptive memory programming. The remaining parts of this chapter explain the fundamentals and types of adaptive memory programming approaches currently in use.

1.3. Intelligent Optimization Fundamentals

Intelligence is the ability to respond to varying conditions, or of changing decisions over time as a function of multiple considerations. Adaptive memory

programming (AMP) suggests that an *adaptive memory* and a *responsive exploration* need to be integral part of a search algorithm. Adaptive memory means ability to selectively remember key elements of previous information stored in the memory while responsive exploration means ability to make strategic choices concerning the direction to follow. This contrast between the adaptive memory programming orientation and the local optimality orientation is significant. Responsive exploration exploits the information desirable from a selected strategy. Such information may be more substantial, even if the selected strategy is in some sense a bad strategy, than the outcome from a good random choice. In an optimization technique that uses memory (such as adaptive memory programming), a bad choice based on strategy provides opportunity to intelligently modify the strategy. Memory-based systems provide some clues as to how one can modify the strategy. Additional features of adaptive memory programming include: the kind of neighbourhood exploration used, intensification and diversification mechanisms, the number of current solutions transported from one iteration to the next, and ability to improve on initial seed solution from an inferior heuristic.

Figure 1.5 shows the classification of the six fundamental features of adaptive memory programming under strategic, tactical, and operational levels. At strategic level, an adaptive memory programming scheme, is driven by *adaptive memory* and a *strategic guidance (exploration)* mechanism. At tactical level, an adaptive memory programming searches over a restricted search-space for

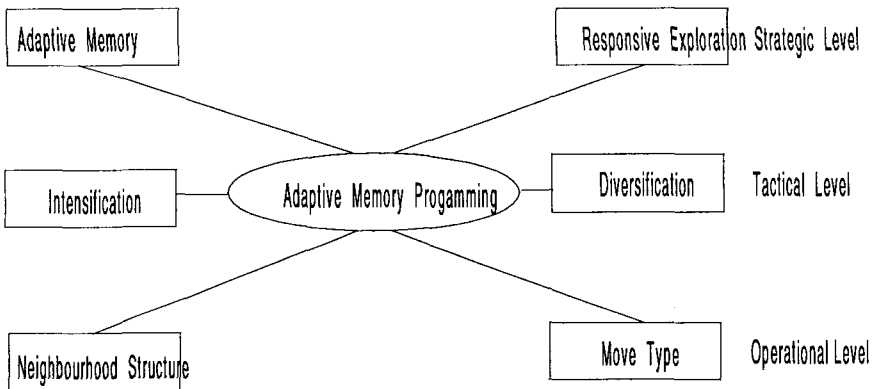


Fig. 1.5. Features and levels of Adaptive Memory Programming.

each iteration known as *neighbourhood*. It should be able to utilize intensification and diversification mechanisms to explore better solutions within the solution space. At operational level, the *move type* determines the number of current solution carried forward from one iteration to the next. It should also be able to consider initial solution from constructive moves. These fundamental features are now discussed in more details.

1.3.1. Adaptive memory

The contrast between the adaptive memory programming orientation and the local optimality orientation is significant. Iterative heuristics that give local optimum are often referred to as ascent methods, descent methods, steepest descent methods, or local search methods. The term “local search” is currently sometimes being used in a very loose way to include search that is not limited to being local. Therefore, care must be taken to understand the context in which the term “local search” is used in the literature.

Memory is no doubt an integral component of any search that qualifies to be called “intelligent”. Adaptive memory depends intimately on four elements of recency, frequency, quality, and influence. Recency-based memory keeps track of solutions attributes that have changed during the recent past. Frequency-based memory provides useful information that facilitates choosing preferred moves. Quality-based memory differentiates the merit of solutions visited during the search. When operationalizing quality-based memory, inducements provide reinforcement actions that drive the algorithm towards good solutions and penalties are provided to discourage action that lead to poor solutions. Influence considers the impact of strategic exploration on both quality and structure.

Three types of memory are commonly used: associative, attributive and explicit. In associative memory structure, given a pair of vectors, an input of one of the vectors should result in the output of the other corresponding vector. Moreover, a noisy, or corrupted, version of one of the input vectors should also recall the corresponding output vector. Associative memory based structures store vectors so that later, if a noisy vector, or an incomplete vector is used as an input, the algorithm will detect the stored vector closest in some sense to the input vector. In attribute memory structure, information regarding solution attributes are recorded as the attributes change in transition from one solution to another. In explicit memory structure, the entire attractive solutions are memorized and used to expand the local search.

1.3.2. *Strategic exploration*

Strategic exploration or choice requires a strategic guidance mechanism. An adaptive memory programming may also modify the neighbourhood of moves in the search space by selectively pruning some neighbours and including others. This approach enlarges a standard neighbourhood that allows moves only among feasible solutions to include unrealizable solutions. The search is then strategically driven beyond the feasibility boundary into the unrealizable region. The search progresses to some depth in the unrealizable solution space and then retracts into the feasible solution space and progresses in the direction of increased feasibility. The process is repeated as the search progresses. The strategic guidance mechanism that drives the search in and out of unrealizable solution spaces requires that a change in the nature of evaluation be made.

1.3.3. *Intensification*

Intensification and diversification strategies constitute the two important components of adaptive memory programming. In the adaptive memory programming framework presented in Fig. 1.5, intensification and diversification strategies are positioned at the tactical level.

Intensification is one of the important components of adaptive memory programming. Intensification strategies are based on modifying choice rules in order to favour move combinations and elite solutions. Intensification strategies may also return to attractive or green regions to search them more thoroughly. This concept agrees with natural phenomenon. During intensification, the neighbours of elite solutions are examined more thoroughly. Two things happen when the neighbours of elite solutions are examined: (1) neighbours to elite solutions are considered using standard move mechanism, (2) neighbours are generated by using modified evaluation or choice strategies that favour the introduction of such components into a current solution.

1.3.4. *Diversification*

Diversification is another one of the important components of adaptive memory programming. In the framework of adaptive memory programming discussed in Fig. 1.5, intensification and diversification are positioned at the tactical level, thus, showing their importance. Diversification favours the search process to

examine unvisited regions in the solution space and generates solutions that differ in various significant ways from those already obtained.

1.3.5. *Neighbourhood*

The neighbourhood N_i for each solution i , is made up of all solutions that can be reached from i in one transition. Adaptive memory programming explores the solution space by repeatedly making moves from one solution to another solution located in the neighbourhood. These moves are performed with an ultimate goal of reaching a good solution by the evaluation of some objective function to be optimized (maximized or minimized). However, contrary to other traditional iterative methods, in adaptive memory programming, the subsequent function evaluated need not be better in solution quality than the current function evaluated, in every iteration.

There are three commonly described neighbourhood structures: exhaustive, randomized, and dynamic. In exhaustive neighbourhood structure, the whole neighbourhood space is searched. Exhaustive searches are impracticable for large-scale combinatorial optimization problems. In randomized neighbourhood structure, not all the neighbours are examined. Here, neighbours are selected at random, and the objective function at that neighbour is evaluated. If the value is the smallest value that has been found so far it will be stored as the minimum value. The variable neighbourhood structure is the result of maintaining a selective history of the state encountered during the search. Operationally, the modified neighbourhood may be shrunk as a subset of the current neighbourhood or may be expanded to include solutions not ordinarily found in the current neighbourhood. The latter case is a description of a dynamic neighbourhood. It is dynamic because the neighbourhood can change according to the history of the search.

1.3.6. *Move type*

There are two types of moves that are commonly considered: non-solution space-based move and solution space-based move. In the former approach, the solution search moves from one current solution to the next after each iteration and uses the properties of the solution for searching intelligently in the solution space. In the latter approach, collective properties of a group

of distinguishable solutions, called solution space, are used for searching intelligently in the solution space.

1.3.7. *Solution from constructive methods*

Constructive methods have already been discussed. As can be seen, their solutions are obtained in a static manner. Constructive methods cannot improve their solution. An adaptive memory programming approach should be able to receive a seed solution from a constructive method and then improve on this. The idea of merging constructive methods and adaptive memory programming in order to refine a seed solution is now popular amongst researchers. It is a good way of expanding the scope of hybrid systems.

1.3.8. *Generic scheme of an adaptive memory programming*

An adaptive memory programming works as follows: a new solution of the combinatorial optimization problem being solved is constructed by first creating an intermediate solution using information in the memory and then this intermediate solution is improved with a local search procedure. Finally, the improved solution is used to update the memory and the process is continued until a stopping criterion is reached. The generic scheme of an adaptive memory programming is given in Fig. 1.6.

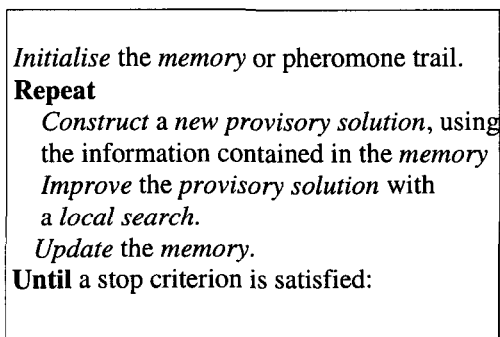


Fig. 1.6. General scheme of an Adaptive Memory Programming.

Now that the fundamental features of adaptive memory programming have been discussed in some details, it becomes a simple matter to classify extant approaches used for solving combinatorial optimization problems.

1.4. Adaptive Memory Programming

There are many optimization techniques that are labeled “intelligent” whereas they are not. The notion of intelligence presupposes that memory exists. It is not enough to have memory. It is how the memory is used that determines whether a system is intelligent or not. We have already defined what intelligence means. It is not sufficient for human beings to remember everything. Over the years we come across a number of events, perhaps in their thousands. Irrelevant events should not be remembered because they do not add value to our lives. However, important events need to be remembered and properly utilized. Suppose that some years ago, you were driving your car and then you were stopped by the police who ask you to produce your driving license and your vehicle particulars. You neither carried your driving license nor your vehicle particulars, nor the photocopies. The police then gave you a ticket for not having the documents in your possession. Of course, this event should be remembered. When next you want to drive out for a leisure trip through a highway where police usually stop cars randomly for some checks, you want to be sure you have your vehicle particulars in your possession. Why? Because you remember that unpleasant incidence when you were served with a ticket when you did not have your vehicle particulars in your possession. Assuming some years back you went into the city centre for shopping and saw some people cleaning their floors with hovers. Or you went to the post office and saw some workers selling stamps and telephone cards across the counters. After a year or two do you need to remember that you saw some cleaners cleaning the floors of their shops or some post office clerks selling stamps and telephone cards over sales counters? The answer is obviously, no. Why? Because remembering these events do not add value in any way. This is essentially the point we are emphasizing in intelligent or emerging optimization techniques. One of the strategic features of adaptive memory programming is that of *adaptive memory*. Adaptive memory means remembering selective events that add value to the goal of the system. The system we are referring to here, is search system. This one key feature disqualifies some search techniques that are labeled “intelligent”. Although it is not the only feature that qualifies

an optimization technique as “intelligent”, it is a highly important one and perhaps the most important. We recapitulate the features that qualify an optimization as being “intelligent”. They are as follows: (1) adaptive memory (2) strategic exploration or choice (3) intensification mechanism, (4) diversification mechanism (5) neighbourhood structure (6) the number of current solutions transported from one iteration to the next (7) and, ability to improve on an initial seed solution from an inferior heuristic. The first two are at the strategic level, the third and fourth are at the tactical level, while the last two are at the operational level.

The past one-decade has witnessed tremendous research works and publications in the area termed “modern search techniques”. There has been proliferation in the use of terminology to describe these search techniques. In the literature, you will come across the terms such as “modern heuristics”, “randomized algorithms”, “local search”, etc., used to describe these recent search techniques that have the power to solve large-scale combinatorial optimization problems. They all seek to introduce mechanisms through which the search techniques escape local optimality. Commonly listed modern optimization techniques include simulated annealing, neural networks, genetic algorithms, tabu search, scatter search, and ant systems.

This book is not intended to be a monograph of optimization search techniques classifications. However, we have laid the foundation for determining to some extent what techniques pass the test of being referred to as adaptive memory programming. Of all the techniques listed only simulated annealing does not incorporate memory in the context we have discussed so far. It is therefore, excluded from the list of adaptive memory programming.

Adaptive memory programming approaches have connections with physical and natural phenomena. Neural networks seek to imitate the human brain. Genetic algorithms mimic chemical reactions in organic compounds. Tabu search derives and exploits a collection of principles of intelligent problem solving. Scatter search follows the principles that underlie the surrogate constraint design. Ant systems simulate the collective contributions of a colony of ants in solving a common problem.

1.4.1. *Explicit memory versus implicit memory*

The adaptive memory classification is an explicit intelligent memory construction. An adaptive memory constructs a new solution by first creating an intermediate solution using information contained in the memory and then

this intermediate solution is improved. The improved solution is then used to update the memory and the process is repeated. Ant systems, tabu search, and neural networks clearly have the features of explicit memory.

In an implicit memory structure, solution combination methods pass on features of selected past solutions to current solutions. These past features are not remembered in a constructive way. Loosely speaking, some good solutions of the past are pushed forward to compete with subsequent solutions. There is an analogy here with sports competition. Think of world sports competition. Athletes compete in their states and the winners are declared. These may be two, three or more. Then there is a national competition in each country. The winners from each state congregate for the national competition. This progresses into international competitions, each time some competitors drop out while others are winning. It is a game of winners. The competitors at the international level are identified with their nations through their jerseys and the records kept by the organizers. It is difficult to trace all the competitors that dropped out. This is because only the best competitors are pushed forward to compete at subsequent level. This simple analogy explains what happens in optimization techniques that have implicit memory structure. Solution combination methods pass on features of selected past solutions to current solution. Genetic algorithm is an example of combinatorial optimization technique that has implicit memory structure.

1.4.2. *Neural networks*

A first adaptive memory programming approach is artificial neural networks (ANNs). Maren *et al.* (1990) define a neural network “as a computational system, either hardware or software, which mimics the computational abilities of biological systems by using a large number of simple, interconnected artificial neurons”. Artificial neurons are simple emulation of biological neurons; they take information from sensor(s) or other artificial neurons, perform very simple operations on this data, and pass the results on to other artificial neurons. The combination of simple processing units and complex connections result in a computing tool that is able to perform high non-linear, multi-input, multi-output function.

ANNs correspond to a massively parallel-distributed processor with a natural propensity for storing experiential knowledge and making it available for future use. ANNs resemble the brain with respect to the capacity of

acquiring knowledge through a learning process and storing it in the inter-neuron connections (Haykin, 1994). Many architectures and configurations of ANNs can be built in order to perform different applications, such as classification, clustering, regression and optimization (Kung, 1993). Artificial neural networks are known to be very powerful in scientific and engineering applications when used to predict, classify or recognize patterns due to the inherent data classification capabilities and massively parallel processing power. Incidentally, they have not been as successful as the best adaptive memory programming techniques from the operations research literature when applied to combinatorial optimization problems. The interest in using neural networks for combinatorial optimization problems began in the mid-1980s with the work of Hopfield and Tank (1985, 1986) on the traveling salesman problem. There are two basic approaches to using neural networks for solving optimization problems.

The first approach is based on statistical physics and includes the Hopfield-Tank and the Mean-Field annealing neural networks algorithms. In the Hopfield-Tank approach the set of all possible solutions of a combinatorial optimization problem is mapped onto the set of all possible states of the network. The network is normally composed of one-layered neurons with fully-weighted connections. An energy function is defined over the set of all possible states and it depends on the weights of the connections of the network. The neural networks algorithm iterates towards a ground state of minimum energy that corresponds to an optimal solution satisfying all constraints of the combinatorial optimizations problem. Due to the feedback characteristics, the Hopfield-Tank model converges to a local minimum. Various modifications have been proposed to alleviate this problem such as the incorporation of simulated annealing in a Boltzman machine and other stochastic models.

The second approach is based on competitive neural networks and includes the Kohonen self-organizing network and deformable (elastic) nets in that neurons are allowed to compete to become active under certain conditions. The aim of the Kohonen model is to find a mapping from a high dimensional input space onto a lower dimensional discrete lattice (usually two) of formal neurons. The mapping tries to reflect the topological neighbourhood relationships among the inputs in the arrangement of the corresponding neurons in the lattice. The above two approaches suggest neural networks as an alternative for solving certain optimization problems as compared to classical optimization techniques and other novel approaches. For information on combinatorial

optimization problems and neural networks, refer to Udo and Gupta (1994), and Garetti and Taisch (1999).

1.4.3. Genetic algorithms

A second adaptive memory programming approach is known as Genetic Algorithms (GAs) (Goldberg, 1989; Davis, 1991). Simply, GAs reach the solution of a problem by generating and modifying a certain number of tentative solutions so as to find the individual (i.e. the solution) best suited to a given environment (i.e. the problem domain). A solution is represented by a string of numbers or characters forming a so-called *string*. A set of strings is referred to as a *solution space*. After a random initialization of a solution space, the algorithm proceeds in an iterative way. It proceeds through: (i) an evaluation of strings by the way of a score function with respect to an objective function of the problem; (ii) a selection of candidates and recombination (iii) a generation of a new solution space from the old one through genetic operators of partial-bits-exchange and partial-bits-exchange. The algorithm stops when a given criterion is reached. Due to the particular way GAs represent the problem solution (sequence of bits), their use in combinatorial problems, such as scheduling, is quite natural (Goldberg, 1989).

1.4.4. Tabu Search

A third adaptive memory programming approach is represented by Tabu Search (TS), proposed by (Glover, 1989, 1990) and (Hansen, 1986) as an effective technique to solve combinatorial optimization problems. The TS method is a adaptive memory programming that shares with the genetic algorithms and ant systems the ability to guide the local search descent method to avoid bad local optima. It uses deterministic acceptance criterion. The method works this way: starting from an initial feasible solution (also a trivial one), a modification of the current solution, which results in another feasible solution, is defined as a *move*. For example, an interchange of two jobs in a sequence can be referred to as a move, even if any reasonable method can be used depending on the problem being studied. All the solutions that can be obtained by a single move from a certain solution are referred to as the neighbourhood of the solution itself. Starting from a feasible solution, all the neighbouring solutions are evaluated and the best one is chosen. Anyway, as a result of

the move a worse solution can be chosen. On the next iteration, a typical greedy method would choose the previous current solution, thus resulting in cycling. Avoidance of such undesirable effect in tabu search is featured by the tabu list. All solutions not allowed to be selected at the current iteration, are inserted in this list. During each iteration, the past current solution becomes tabu and the oldest move in the tabu-list leaves the list itself. A neighbouring solution is considered forbidden and deemed not admissible if it has attributes on the tabu list. Storing attributes rather than the complete solutions may cause some non-tabu solutions to be wrongly prevented and aspiration criteria are normally used to correct such errors. Therefore a move remains in the tabu list as many moves as the tabu list size. Sometimes, a tabu move can assume the highest value of the objective function ever obtained. In that case, TS recalls an aspiration criteria to override the tabu restrictions and selects anyway the move as the current solution. To produce good results, intensification and diversification strategies are also employed. The former strategy is used to perform a thorough search in good regions while the latter strategy is attempted to consider solutions in a broad area of the search. Different stopping criteria can be developed, even if the most common is a fixed number of iterations without any improvement in the best move.

1.4.5. *Ant systems*

A fourth adaptive memory programming approach is known as ant colony optimization (ACO) introduced by Dorigo, Maniezzo and Colorni (1996). They studied artificial systems that take inspiration from the behaviour of real ant colonies and which are used to solve discrete optimization problems. The first ACO system was called Ant System (AS), which is the result of a research on computational intelligence approaches to combinatorial optimization problems. In AS, an ant represents a single artificial agent and the set of artificial ants cooperate in the search for an optimal solution by exchanging information in the form of a pheromone trail. Since the introduction of AS, there have been a number of variants such AS-Q that uses the Q-learning methodology, hybrid ant system (HAS) and fast ant (FANT) system.

Neural networks, genetic algorithms, ant systems and tabu search all belong to the class of Adaptive memory programming. In fact, starting from a solution or set of solutions, they proceed applying special algorithms that make a limited change in the solution itself. Therefore, the new solution is not too

far (local) from the previous one. Such a way to operate has been utilized by many authors to refine a first optimal solution found using a different method. Adaptive memory programming have been found to have the power to solve highly constrained, practical, real-life, problems (Dorn *et al.*, 1996).

1.5. Hybrid Systems

Adaptive memory programming approaches have proven powerful in finding high quality solutions to many real world intractable problems. Therefore, over the years, researchers have attempted to combine the best features from different adaptive memory programming approaches to derive more powerful hybrid implementations. It is only natural to do this. Combining the best features of different systems will give a new system that is superior to the individual systems from which these features are derived. Hence, adaptive memory programming continue to prove superior to other known heuristics. It is expected that more serial and parallel implementations based on hybrids of adaptive memory programming with each other or with classical operational research techniques will be on the increase.

In this section, we describe how hybrid adaptive memory programming works, and then we give a table of some known hybrid systems that have been implemented by researchers. Let us describe the Genetic-Tabu Search Hybrid (GTSH) proposed by Fleurent and Ferland (1994). By so doing, readers will easily see the benefit of hybridization.

The GTSH method can be presented as a genetic algorithm, using a special partial bits exchange operator specially designed for permutations (due to Tate and Smith, 1995) and a random bits exchange operator that consists in performing $4n$ steps of a tabu search (due to Taillard, 1991). Let us now describe the memory, its implementation, constructive generation of a provisory solution, improvement via tabu search, and memory update in GTSH.

The memory of GTSH consists of 100 solutions or permutations. Initially, these solutions are obtained by 100 tabu search runs performing $4n$ iterations each and starting with random initial solutions.

The provisory solution is built by first selecting two permutations from the solution space. The probability of selecting the k th worst solution of the solution space is $2k/(s - 1)$, where s is the size of the solution space. The partial bits-exchange of Tate and Smith is then applied to the selected solutions

to produce the provisory solution. The elements common in both selected permutations are copied in the new solution simultaneously. Other elements are chosen randomly from the selected permutations, at the same position, if not already chosen. The unfilled positions are chosen randomly in order to complete the solution.

The improvement procedure is the tabu search of Taillard (1991) that runs for $4n$ iterations, starting from the provisory solution. The new solution is inserted in the memory and the worst solution is removed from the memory.

It becomes evident that the description of GTSH is different from the standard genetic algorithm description. But genetic algorithms and tabu search can be joined in a more fundamental way. The introduction of tabu search as an improvement procedure makes GTSH a hybrid system. In traditional genetic algorithm implementation, there is no such improvement procedure other than the generic operators such as partial bits-exchange and random

Table 1.3. List of some hybrid systems.

Year	Investigators	Description
GENETIC ALGORITHM AND LOCAL SEARCH		
1983	Goldberg	Suggested hybridizing genetic algorithm with problem-specific search techniques.
1983	Goldberg	Hybridized genetic algorithm with local search known as G-bit improvement (gradient-like-bitwise improvement)
1997	Taillard and Gambardella	Genetic Algorithm-Descent Method hybridization (GDH)
GENETIC ALGORITHM AND TABU SEARCH		
1996	Fluerent and Ferland	Tabu search and genetic algorithm hybrid. Details already discussed in this chapter
NEURAL NETWORKS AND TABU SEARCH		
1989	de Werra and Hertz	Applied hybrid neural network and tabu search to visual pattern recognition, with increased speed.
1991	Beyer and Ogier	Applied hybrid neural network and tabu search to non convex optimization.

bits-exchange that are embedded in genetic algorithms. The introduction of explicit memory, constructive generation of a provisory solution, improvement via tabu search, and memory update, qualify GTSH as a hybrid system. Normally, a hybrid system will yield better solution than the subsystems that constitute it. Therefore, one expects GTSH to perform better than genetic algorithm and tabu search, separately. Table 1.3 gives a list of some of numerous implementations of hybrid system. For readers who are interested in pursuing the matter further, the bibliography compilation of Osman and Laporte (1996) may be consulted.

Hybridization has taken several forms in recent years. For example, researchers now hybridize adaptive memory programming approaches such as genetic algorithm, neural networks, tabu search, and ant systems, with other good heuristics to generate initial solutions. By so doing, the global solutions obtained by adaptive memory programming are drastically improved. How do we explain this improvement? It is simple. Most adaptive memory programming approaches generate initial solution randomly. Random initial solution may not satisfy the constraint that the performance measure has to be optimized. Therefore, utilizing a good heuristic for generating initial

Table 1.4. Some current hybrid extensions to scheduling problems.

Year	Investigators	Description
TABU SEARCH		
1989	Widmer and Hertz	Hybridized tabu search and an analogy to TSP for makespan performance measure.
1990	Taillard	Hybridized tabu search and an improved NEH algorithm for makespan performance measure.
GENETIC ALGORITHMS		
1995	Chen, Vempati, and Aljaber	Hybridized genetic algorithm and CDS along with rapid access approach for makespan performance measure.
ANT SYSTEMS		
2000a	Onwubolu	Hybridized ant system and NEH;
2000b	Onwubolu	Hybridized ant system and curtailed insertion method

solution is known to yield better quality solution. Table 1.4 shows some of current hybridization of adaptive memory programming with good heuristics for generating initial solutions for scheduling problems.

1.6. Summary

In this first chapter, both the conventional optimization techniques for solving combinatorial problems, and production planning and control, are presented.

Production planning decisions span the *planning schedule* and the *capacity schedule* spectrum. The planning schedule consists of demand forecasting, aggregate production planning, master production schedule, and material requirement planning. The capacity schedule consists of resource capacity planning, rough-cut capacity planning, and capacity requirement planning. Production control decisions span the *planning execution* and the *capacity execution*. The planning-execution consists of operation sequencing, scheduling, order priorities, stock picking, and capacity control. The capacity execution consists of tool control, machine control, factory order control, inventory control, and capacity control.

The conventional optimization techniques for solving combinatorial problems fall into two classes: optimization algorithms and approximate algorithms otherwise known as heuristics. The methods that fall under the first class are attractive if the problems solved are of small size and if optimality must be achieved. These methods fail to perform for large size problems, typically encountered in real life situations. The reason for this failure is that of computational time involved, for which most computers cannot accommodate. The methods that fall under the second class are also attractive if all that is needed is a quick solution, without attention to very good quality. Between these extremes naturally lies a method that finds near optimal solutions in reasonable computational time for practical, real life problems. Adaptive memory programming fills this gap. We have presented the features of adaptive memory programming. These are generally accepted as the yardstick for determining whether or not a method qualifies as being intelligent. Neural networks, genetic algorithms, tabu search, and ant systems are presented as being adaptive memory programming approaches. They qualify as Emerging Optimization Techniques. These are briefly discussed. Hybrid systems are presented as extensions of adaptive memory programming fundamentals. This chapter therefore, sets the scene for what follows in the

remaining parts of the book. The details of each of these are discussed in the following chapters. You will notice that simulated annealing is not classified as belonging to adaptive memory programming. This is because it does not have memory since adaptive memory is paramount in classifying an approach as belonging to adaptive memory programming.

Exercises

Production Planning and Control

1. What activities constitute *planning-schedule* aspect of the production planning and control system?
2. What activities constitute *capacity schedule* aspect of the production planning and control system?
3. What activities constitute *planning-execution* aspect of the production planning and control system?
4. What activities constitute *planning-execution* aspect of the production planning and control system?
5. What is the role of production planning and control in integrated manufacturing?

Adaptive memory programming

1. Discuss the disadvantages of optimization algorithms such as *branch-and-bound* and *A** in solving combinatorial optimization problems?
2. Are constructive methods good compared to adaptive memory programming approaches in terms of solution quality? Discuss.
3. Discuss the interpretation of the six adaptive memory-programming dimensions: adaptive memory, strategic exploration, and intensification/diversification, move type, and acceptance of seed solutions from constructive methods.
4. What are the advantages of memory-based structures over memoryless-based structures in designing schemes for solving combinatorial optimization problems?

References

Production Planning and Control

- Filho, O. S. S., 1999, An aggregate production planning model with demand under uncertainty, *Production Planning and Control*, **10**(8) 745–756.
- Garetti, M. and Taisch, M., Neural networks in production planning and control, 1999, *Production Planning and Control*, **10**(4) 324–339.

- Gelders, L. F. and van Wassenhove, L. N., 1981, Production planning: A review. *European Journal of Operational Research*, **7** 101–110.
- Melnyk, S. A., and Denzler, 1996, *Operations Management: A Value-Driven Approach* (Irwin Publishers: Boston).
- Nawaz, M., Enscore, E. E., and Ham, I., 1983, A heuristic algorithm for the m -machine, n -job flow shop sequencing problem, *OMEGA The International Journal of Management Science*, **11**(1) 91–95.
- Onwubolu, G. C. and Mhlanga, S., 1997, Development of a computer-aided production management course in industrial engineering curriculum, *Production Planning and Control*, **8**(3) 297–306.
- Vollmann, T. E., Berry, W. L., and Whybark, D. C., 1991, *Manufacturing Planning and Control System* (Homewood, IL: Irwin).
- Vonderembse, M. A. and White, G. P., 1988, *Operations Management: Concepts, Methods and Strategies*, 1st edition (West Publishing, St. Paul, MN).
- Wild, R., 1992, *Essentials of Production and Operations Management*, 3rd edition (Casell Educational, London).

Adaptive Memory Programming

- Beyer, D. A. and Ogier, R. G., 1991, Tabu learning. A neural network search method for solving non-convex optimization problems, in *Proceedings of the International Conference on Neural Networks* (IEEE Computer Society Press, Los Alamitos, California), 953.
- Chen, C.-L., Vempati, V. S., and Aljaber, N., 1995, An application of genetic algorithms for flow shop problems, *European Journal of Operational Research*, **80** 389–396.
- Davis, L., 1991, *Handbook of Genetic Algorithms* (VNR computer Library, NY).
- De Werra, D. and Hertz, A., 1989, Tabu search techniques. A tutorial and an application to neural networks, *OR Spectrum*, **11** 131.
- Dorigo, M., Maniezzo, V., and Coloni, A., 1996, The ant system. Optimization by a colony of cooperating agents, *IEEE Transaction on Systems, Man and Cybernetics-Part B*, **26**(1) 29–41.
- Dorn, J., Girsch, M., Skele, G., and Slany, W., 1996, Comparison of iterative improvement techniques for schedule optimization. *European Journal of Operations Research*, **94** 349–361.
- Fleurent C. and Ferland, J., 1994, Genetic hybrids for the quadratic assignment problem, *DIMACS Series in Math. Theoretical Computer Science*, **16** 190–206.
- Glover, F., 1989, Tabu search, Part I, *ORSA Journal on Computing*, **1**(3) 190–206.
- Glover, F., 1990, Tabu search, Part II, *ORSA Journal on Computing*, **2**(1) 4–32.
- Hansen, P., 1986, The steepest ascent mildest descent heuristic for combinatorial programming, Presented at the *Congress on Numerical Method in Combinatorial Optimization*, Capri, Italy.
- Goldberg, D. E., 1989, *Genetic Algorithm in Search, Optimization and Machine Learning*, Addison Wesley, Workingham, England.

- Haykin, S., 1994, *Neural Networks. A Comprehensive Foundation*. Macmillan College Publishing Company, NY.
- Hopfield, J. J. and Tank, D. W., 1985, Neural computation of decision in optimization problems, *Biological Cybernetics*, **52** 141–152.
- Hopfield, J. J. and Tank, D. W., 1986, Computing with neural circuits: A model. *Science*, **233** 625–633.
- Kung S. Y., 1993, *Digital Neural Networks*. Prentice Hall, Englewood Cliffs, NJ.
- Maren, A., Harston, C., and Pap, R., 1990, *Handbook of Neural Computing Applications*. Academic Press, San Diego, CA.
- Onwubolu, G. C., 2000a, Ants can schedule, *Industrial Engineering Research Conference*, Cleveland, Ohio, USA, May 21–23, 2000.
- Onwubolu, G. C., 2000b, Ant system approach for the flow shop problem with bi-criteria of makespan and total flow time minimization, *From Ant Colonies to Artificial Ants: 2nd International Workshop on Ants Algorithms*, Brussels, Belgium, September 8–9, 2000, 96–99.
- Osman, I. H. and Laporte, G., Metaheuristics: A bibliography, *Annals of Operations Research*, **63** 513–623.
- Taillard, E., 1990, Some efficient heuristic methods for the flow shop sequencing problem, *European Journal of Operational Research*, **47** 65–74.
- Taillard, E., 1991, Robust tabu search for the quadratic assignment problem, *Parallel Computing*, **17** 443–455.
- Taillard, E. and Gambardella, L. M., 1997, *Adaptive Memories for the Quadratic Assignment Problem*, IDSIA-87-97 Technical Report, Switzerland.
- Tate, D. E. and Smith, A. E., 1995, A genetic approach to the quadratic assignment problem, *Computers and Operational Research*, **1** 855–865.
- Udo, G. J. and Gupta, Y. P., 1994, Applications of neural networks in manufacturing management systems, *Production Planning and Control*, **5** 258.
- Widmer, M. and Hertz, A., 1989, A new heuristic method for the flow shop sequencing problem, *European Journal of Operational Research*, **41** 186–193.