CHAPTER 6

APPLYING THE EQUAL PILES APPROACH TO BALANCE ASSEMBLY LINE

Excellence is never granted to man
but as the reward for labour.

- Sir Joshua Reynolds

Keywords: assembly line design, equal piles, grouping genetic algorithm, line balancing, logical line layout, multiple objective.

1. Introduction

Assembly line (AL) are the most commonly used method in mass production environments because they enable assembly of complex products by workers with limited training. The main objective of assembly systems designers is to increase the efficiency of the line by maximising the ratio between throughput and required costs. Thus, assembly line design is a problem of considerable industrial importance.

Each company taking the decision that assembly is a strategic core activity, is faced with the question of how to utilise and develop the assembly system in the most efficient way in order to become successful. The assembly activities performed within the assembly system not only determine the final qualities of the products, but also affect time-to-market, delivery, etc. Emphasising the way in which an assembly system is designed will, therefore, control its efficiency and quality. For manual

assembly line the most interesting performance index is stations workload's *balancing*. The problem is known as the 'assembly line balancing'(ALB).

In this chapter we focus on single and multi-product assembly line balancing problems. Section 2 is dedicated to works on the assembly line balancing. Section 3 formally describes the new method proposed which is called 'equal piles' for assembly line. In section 4, is introduced the multi-product assembly line balancing problem. We finish by drawing some conclusions and propose further works.

2. State of the art

Related assembly line design problems and issues are characterised in the literature as the ALB problem, which usually refers to single product assembly line. Most papers deal with a single objective: to maximise the efficiency of the assembly line through minimisation of idle time. The simplest version of the problem is the well-known bin packing problem (BPP), where the aim is to pack uni-dimensional '*objects*'of various sizes into as few identical 'containers'of a given capacity as possible. Assuming that the durations of the tasks to be performed are fixed, they can be taken for the unique dimension of the 'objects'. Then casting the cycle time as the capacity of the bins, the problem becomes how to distribute operations among stations in such a way that no station overflows the cycle time and the minimum number of stations is used. Thus, the BPP leads to a minimisation of the number of stations along the line. Note however that this simple formulation does not take into account the precedence constraints between operations, which amounts to assume that a product can be assembled by any succession of operations. Simple assembly line balancing (SALB) and its varieties are related to various assignment, sequencing and grouping problems. Many exact methods and approximated approaches are cited in literature.

Since the ALB problem is known to be computationally difficult, exact solution of large instances will often require an excessive amount of time. In such cases, it may be more appropriate to use a heuristic or an approximated method. Even for small size instances, a good heuristic can be also used as initial solution for enumerative methods.

2.1. Exact methods

We here consider only the most famous versions of the SALBP (a detailed description of the other versions is done in section 5.6 of Chapter 3). Many kinds of constraints and objectives are cited as well as some modifications of the mathematical programming formulations. Several approaches for determining *lower bounds* on the number of stations *n* in the case of SALBP-1 (the cycle time in the case of SALBP-2) are proposed in the literature. The lower bounds are obtained by solving problems which are derived from the considered problem by omitting or relaxing constraints. Most of these techniques fall into two categories which are: dynamic programming and branch and bound methods.

A good introduction to optimal approaches of the ALB problem can be found in (Baybars, 1986). A good survey on exact methods for the ALB problem and its varieties can be found in (Scholl, 1999).

2.1.1. Dynamic programming

The dynamic programming (DP) method is applied to most combinatorial optimisation problems. It involves the optimisation of multi-stage decision procedures. A given problem is divided into sub-problems which are sequentially solved until the initial problem is finally solved. States at a particular stage s are transformed to states at the subsequent stage $s+1$ by a decision. The generation of states is described by *transformation functions* which depend on the current state and the decision taken. A sequence of decisions, which transforms a state at a stage s to a stage $s'>s$, is called *policy*. The DP is a solving approach rather than a technique. The following approaches are linked to this technique.

Salveson (Salveson, 1955) was the first to empirically address the ALB problem. He described it as a set of precedence constraints between tasks that must be adhered to. Salveson also recognised the ALB problem as a combinatorial one. He formulated the SALB problem as a linear programming (LP) problem including all possible combination of station assignments. His model, by definition, can result in spliting tasks and may generate infeasible solutions. Bowman (Bowman, 1960) was the first to provide a 'non-divisibility' constraint, by changing the LP formulation to zero-one integer programming. Due to the high number of variables the method requires, even for small problems, the work of both Salveson and Bowman is impractical to find optimal solutions. Patterson (Patterson, 1975) used the integer programming search technique. His method was more computationally efficient than some of the earlier ones because of some unnecessary variables were eliminated from the formulation.

The SALBP can be represented by a *tree* in which each path corresponds to a feasible solution with each arc representing a station. Jackson (Jackson, 1956) was the first to propose an algorithm for SALBP-1 using the notion of a tree. A station is termed maximal if no task can be assigned to it without violating the precedence and the cycle time constraints. The method starts by generating all feasible assignments to the first station (one of these feasible assignments will obviously be part of the optimal solution). Then it generates all feasible assignments to the second station, given the first station assignments. Then for each first-second station combination, all feasible solutions are constructed for the third station, etc. The process is repeated, each time adding one station. Jackson uses dominance arguments to eliminate inferior feasible assignments: after generating all feasible assignments, these sets are compared to eliminate subsets that contain exactly the same tasks as other subsets and to eliminate the dominated subsets. A set is said to be dominated if it contains fewer tasks than another subset and does not contain any tasks that are not also in the other subset. Jackson's method does not require the generation of all feasible sequences but still uses exhaustive search of all remaining possibilities.

Peng (Peng, 1991) attempted to take advantage of the additional constraints to obtain an exact solution. However, the inherently combinatorial nature of the problem seems fatal even to his sophisticated DP techniques, especially when the precedence constraints are loose.

Agnetis (Agnetis, 1997) addressed the problem of assigning and sequencing tasks to stations with the objective of minimising the inventory costs or the completion time of a set of identical jobs. The method is limited to products having precedence graph similar to a *comb*. They consider the standard (tasks and their pre-processing tasks are grouped on the same station) and the look-ahead (tasks and their pre-processing tasks are not necessarily grouped on the same station) assignment policies. Three types of sequencing policies are proposed where the main difference is the order in which the tasks assigned to a station are executed. The method is limited to a single product and it is based on the DP technique.

Gutjahr and Nemhauser (Gutjahr, 1964) defined a *feasible subset* as a subset that can be processed without prior completion of any other tasks and in any order that satisfies the precedence relations. Also, they defined a *feasible (sub)sequence* as a subsequence of tasks that can be processed in the indicated order without prior completion of any other tasks. Thus, the later is necessarily an ordered set, unlike the former. The authors used the concept of 'feasible subsets'. The method is a *shortest path* based network technique. It begins by generating a directed network based on the precedence constraints. The nodes correspond to feasible subsets, with the source node being the null set and the sink node being the set of all tasks. The 'time' t_i associated with node i is simply the sum of the process times of the tasks represented by the node. Let $S(i)$ denote the subset of tasks represented by node i. An arc (i, j) is in the directed network if and only if $S(i)$ is a subset of $S(j)$ and $t_j - t_i = C$. Thus, an arc (i, j) corresponds to a station assignment of tasks to $S(j)$. As a result of this, there is a one to one correspondence between source-sink paths and feasible assignments. Thus, finding the shortest path in this directed network is equivalent to finding the minimum number of stations required. While the problem of finding the shortest path is a relatively easy one (Sedgewick, 1984), the number of nodes in the network grows exponentially with the size of the instance, making it unusable for large problems.

2.1.2. Branch and bound

The branch and bound (B&B) algorithm consists of two main components the *branching* and the *bounding*. Additional, *dominance* and *reduction rules* are used to reduce the 'solution effort'. The initial solution is developed into several sub-problems (branching). A multi-level enumeration (descending nodes, the first level consists of the root node) are constructed by continuously developing such sub-problems. Sub-problems for which the optimal solution is already known need not to be branched and are referred to as *leaf* nodes. The same term is used for nodes which are excluded from further consideration (because they cannot lead to an optimal solution). A path from the root node to any other node of the tree is called a *branch*. B&B algorithms

are usually subdivided with respect to the sequence in which the nodes of the enumeration tree are generated and branched:

- The depth-first-search, a single branch of the tree is developed until a leaf node is reached. All the descending nodes may be generated and sorted by a priority rule (according to increasing lower bound values). The one with the highest priority is branched first.
- A minimal-lower-bound strategy selects a not yet developed node which has the minimum value of a lower bound from a candidate list. This node is completely branched by construction of all its descending nodes.

Bounding is applied to reduce the size of the enumeration trees. This is achieved by computing lower bounds (the cycle time in the case of SALBP-2), at least necessary for a feasible solution, in each node. An optimal solution is found if the 'global' lower bound is found. The *dominance* is achieved by identifying dominance relationships between sub-problems. It is based on proprieties of partial solutions and reduced problems, which allow the conclusion that the solution of a sub-problem cannot be better than that of another one. The *reduction rule* tries to modify the problem data, in order to deal with special constraints.

Wee and Magazine (Wee, 1982) showed that SALBP-1 is NP-hard since it is a special case of the BPP which is known to be NP-hard. The authors proposed to solve the problem by means of the generalised bin packing heuristics. A B&B method similar to the one used for the BBP is used. The authors also proposed techniques used in the BPP like the first-fit-decreasing (FFD) rule and the immediate-update FFD (IUFFD) to the ALBP. In the FFD rule method, the tasks are listed in a non-increasing order of process time. In the case of IUFFD, the set of available tasks are immediately updated and arranged in non-increasing order of process time. In general, these two heuristics give good results.

Hoffmann (Hoffmann, 1992) proposed a B&B algorithm which does implicit complete enumeration (ICE) and takes advantage of the theoretical minimum total *idle time* concept. The difference between the cycle time and the sum of the task' process time of a station is referred to as a slack time for a given station. Minimising the number of stations is equivalent to minimise the sum of slack time of stations.

Scholl (Scholl, 1997) proposed a B&B algorithm based on the main ideas of 'fast algorithm for balancing line effectively' FABLE (Johnson, 1988) and EUREKA (Hoffmann, 1992) called 'simple assembly line balancing optimisation method' type 1 (SALOME-1). The method integrates and extends the most powerful components of both former algorithms. Its main characteristic is a branching strategy (local lower bound method) and a bi-directional branching rule as well as dominance and reduction rules. In order to deal with the SALBP-2, the author used a search method involving the repetitive application of SALBP-1 procedures. The meta-heuristic 'tabu search' method is used to overcome local optima. A detailed description of this method is given in (Scholl, 1999).

Since exacts methods are time consuming, many methods use distributed systems to speed-up search. Bock (Bock, 1997) introduced a *distributed fault-tolerant* algorithm for SALBP. He parallelised the B&B procedure (SALOME-1) for SALBP-1. The author presented a search-tree algorithm using more than 1000 transputer-network systems. Due to the use of the fault tolerant concept, if some node (processor) fails the algorithm can go on working without loosing any solution, which is very useful in a network of many personal-computers. Many hard instances for the ALBP were solved for the first time. The fact of using such parallel technique shows how exact methods are *very time consuming* even with speedy computers.

2.1.3. Graph search technique

Johnson (Johnson, 1988) proposed a depth-first-search method called 'fast algorithm for balancing line effectively' (FABLE). Sub-problems are constructed by adding an assignable task to the currently considered station *k* (starting with station 1). If no such tasks exists, the current station load is maximal, and the consecutive station *k+1* is opened. In order to avoid unnecessary permutations of tasks inside stations, loads are constructed as follows. The search method is based on the task numbers. Initially, all tasks are not marked. In each of the *n* iterations *i=1,..n,* one non marked task with largest process time which has no predecessor or only marked predecessors gets the number *i* and is marked. Ties are broken in favour of tasks with the largest number of immediate successors. Second-level ties in favour of the smallest original task number. The method works as follows: whenever a station is opened, the task with the smallest number among the assignable tasks is added. Any further tasks in the station must have a larger number than the task assigned in the ancestor node. After assigning all tasks, a feasible solution is obtained. Now, the current branch is traced back by removing tasks assignments until an alternative branch can be followed, i.e., the latest removed task can be replaced by another task which is assignable to the current station and has a larger number.

2.2. Approximated methods

Talbot et al. (Talbot, 1986) gave a review of line balancing heuristics. They divided heuristics for ALBP into four categories called: 'single-pass', 'composite', 'backtracking' and 'time trapped optimising' approaches. In the first category, task assignment is based upon a single attribute of each assembly task, while decision rules of the second category are a composite of single-pass decision rules. It contains approaches that consist of selecting the best solution determined by the application of several single-pass methods. Methods belonging to the third category are those which, by backtracking, attempt to improve a previously obtained solution. Finally, the fourth category is composed of optimum-seeking approaches with an arbitrary limited amount of computation time.

In this section we will consider some well-known heuristics for the problem. These methods are divided into simple heuristics and metaheuristics.

2.2.1. Simple heuristic methods

Since the ALB problem falls into the NP-hard class of combinatorial optimisation problems, numerous research efforts have been directed towards the development of computer efficient approximation algorithms or heuristics (Ghosh, 1989).

One of the first proposed heuristic was the ranked positional weight (RPW) (Helgeson, 1961). The main idea behind the RPW 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 the sum of the task times of the successor tasks. In the RPW method, the sum of the task process time and the process times of the successor tasks is defined 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 stations if (1) all predecessors of the task have already been assigned and (2) the task fits in the remaining time on the station. If a task does not fit in the remaining time on a station it is skipped and the next task in the ranked order is selected. If no task fits in the station, the station is closed and a new station 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 station. The process is repeated until all tasks are assigned. Since the weight of an element depends on the process time of its predecessors, the algorithm tends to treat the elements near the end of precedence graph first. This procedure is quite simple, it can deal with user's preferences but gives good results only for simple problems.

The 'reversed ranked positional weight' (RRPW) is the RPW method applied to the problem with the reversed precedence constraints. After a balance if found for the reversed problem, the problem is 'un-reversed' to get a balance solution for the original problem (Helgeson, 1961).

Kilbridge and Wester (Kilbridge, 1961) proposed a heuristic based on the cumulative performance times. First the 'layers' are identified in the precedence graph. Tasks with no predecessors are in the first layer. Tasks that are preceded directly by tasks in layer *i* are placed in layer *i+1*, etc. Then, it computes the time for each layer (sum of process time of tasks belonging to a given layer), as well as the cumulative performance time of each layer (sum of a its time and time of the layers it succeeds). Moving from the left to the right in units of a layer, it finds the layer that just fills or overfills the station. It begins by the first layer and go in order through the remaining layers. Tasks of a given layer are assigned to a given station unless the precedence constraints will be violated or the cycle time will be exceeded, otherwise they are assigned to the next station and so on. The method is a search algorithm which is restricted to looking one layer behind and one layer ahead in filling the stations. The heuristic is easy to implement and gives good solutions for dense and uniform graphs. The more the graph contains sprays less the method yields good balancing.

Moodie and Young (Moodie, 1965) presented a modified formulation of ALB problem that includes task time variability. Tasks times are considered to be independent normally distributed with known mean and variance. Their standard

heuristic places tasks into stations starting with tasks having the longest processing time. A task cannot be placed into a station unless all of its immediate predecessors have been already assigned. The shortest time algorithm places tasks into stations according to the shortest processing time.

Gaither (Gaither, 1996) proposed the 'incremental utilisation' heuristic. At each iteration it places a task in a station according to precedence relationships, and a list is constructed with all tasks which are eligible for immediate assignment to a station. A task is selected from a list based upon some rules or heuristics. A task is not added to a station unless its addition results in an increase in the station utilisation–hence the name 'incremental utilisation'.

While the RPW heuristic and its variants give reasonably good solutions, it only gives a single solution. Often objectives other than the minimum number of stations may looked for. This why it is important to obtain several solutions. One way to generate several solutions is to randomly select a feasible task from the available tasks instead of selecting the 'best' task according to some criterion. This is called the random priority. The 'computerised method for sequencing operations on assembly line' (COMSOAL) method, which was introduced by Arcus (Arcus, 1966) uses this strategy. At each stage the set of tasks that can be feasibly assigned and which fit in the remaining time on the current station is determined. The task to be assigned is then randomly selected (each task has the same probability of being selected). The solution which has the lowest level of idle time is then selected.

Hoffmann (Hoffmann, 1963) proposed a heuristic based on a method for generating permutations using a precedence matrix. The procedure works as follows. From the available tasks, a subset is selected such that the current station is loaded as much as possible. The procedure is repeated until all tasks are assigned. Note that such a procedure tends to concentrate tasks either at the first few stations or the last few stations depending on whether a forward or reverse problem is solved.
The author also combined in two ways his approaches and implemented them in a program called 'EUREKA'. The first approach is to use the heuristic first and if it fails to find a theoretically optimal solution, to resort to the implicit complete enumeration (ICE) procedure (Hoffmann, 1992). The second approach is to first use the ICE procedure for a short time interval and if it is not successful, to then switch to the heuristic. EUREKA uses a single simply calculated bounding criteria and an efficient permutation generating scheme to achieve good results.

Boctor (Boctor, 1995) proposed a composite heuristic method to solve SALBP-1. The general framework of this method is similar to most single-pass and composite heuristics. To select the task to be assigned to a given station, this method uses a prioritising scheme composed of four decision rules. The author defined as 'severe task' a task having a processing time greater than or equal to one half of the cycle time. A task is said to be a 'subsequent candidate of task *i*', if it remains or becomes schedulable after assigning *i* to the current station. Among the schedulable tasks, it selects the following: (1) The task having a duration equal to the remaining time. If there is no such task, it uses the next rule. To break ties, it assigns the task with

largest number of 'subsequent candidates'. (2) The 'severe task' having the largest number of 'subsequent candidates'. If there are no severe tasks, use the next rule. In the case of a tie, choose the task with the longest processing time. (3) The combination of two tasks having a duration equal to the remaining time. If there is no such combination it uses the next rule. As a tie breaking rule, it uses the largest number of 'subsequent candidates'. (4) The task having the largest number of 'subsequent candidates'. To break ties it chooses the task having the greatest number of 'severe' immediate successors and if the tie persists, it assigns the task with the longest processing time. Both forward and reverse approaches are tried and the best solutions of the two procedures are retained.

On the side of interactive and iterative methods, Parça (Praça, 1999) proposed an architecture based on multi-agent simulation (called SimBa). The system helps in balancing and distributing the human resources in manual assembly line. The system is composed of a balancing and simulation modules. The *balancing* module which implements three heuristic rules from (Boctor, 1995) and (Helgeson, 1961), allows the user to obtain different line configurations. The *simulation* module is used to evaluate the performance of the proposed configurations. The aim of agent design is to create programs which interact *intelligently* with their environment. The multi-agent system is introduced to simulate the human behaviour in a given AL.

Rachamadugu (Rachamadugu, 1991) investigated the levelling of workloads across stations which is an important problem especially in manual AL. Indeed, uneven assignments are viewed as inherently unfair and usually call for some kind of compensatory actions from management such as differential pay. A *mean absolute deviation* of workloads criteria is used as an objective function to distribute the workload as evenly as possible. The author identified the notion of dominance between solutions and incorporated it into an iterative heuristic. Given any heuristic or optimal solution to a SALBP-1 or SALBP-2, the levelling procedure reassigns tasks among the stations to smooth the workload across stations. It is a kind of *local* optimisation used after the global one. The proposed method yields better allocation of work in the case of SALBP-1 and SALBP-2.

Süer (Süer, 1998) proposed a 3-phase methodology to deal with assembly lines having high production volume. The objective is to determine the number of assembly lines (in parallel) with minimum total manpower. In the *assembly line balancing phase*, tasks are grouped into stations for various configurations (1-station, 2-station, ... , n-station; where *n* is the number of tasks). Thereafter, the *parallel station phase* determines the number of operators that should be assigned to each station (parallel stations). The *parallel lines phase*, determines the number of lines and their configuration with the objective of minimising the total number of operators.

Shtub (Shtub, 1990) pointed out the gap existing between literature dealing with the ALBP and technical documentation used by engineers to describe assembly operations. Comparing the precedence graph (PG) to the assembly chart, it is obvious that PG which presents tasks by their times and precedence relations does not fully describes the relations between a task and the subassemblies on which this

task is performed. An ALB method based only on PG is likely to generate solutions that minimise idle time by loading each station with tasks performed on several subassemblies. Such solutions are in contradiction with the very basic principles of improving work methods and enriching employee jobs. The author proposed two ways to deal with the relationship between a task and the subassemblies on which the task is performed. The first approach aims to minimise the number of subassemblies handled by each station. The second approach aims to constraint the number of subassemblies by each station. The method is applied to SALBP-1 and SALBP-2.

In spite of the existence of many commercial software of 'balancing', less than 10% of companies use such software. The reason is that industrial balancing problems present most of the time, particularities which are not always considered by the standard software. Lapierre (Lapierre, 1999) presented the COMSOAL algorithm (Arcus, 1965) programmed on the software package Microsoft ACCESS97. The author modified the COMSOAL algorithm in order to deal with constraints such as the position (rear, front, centre, etc.) and the level (high and low) of tasks. Thus, the method aims to avoid grouping tasks having different levels on the same station. Tasks requiring heavy equipment or parts must not be assigned to the same station. The approach is very interesting, however, we think that the ALBP is quite difficult to be solved by a simple 'spreadsheet' software. We believe that the simple version of the ALB is quite difficult to be solved by such a method. Anyway, the author recommends using metaheuristic to deal with ALBP.

Many heuristics were proposed in literature and use different criteria (Talbot, 1986). Many proposed heuristics are a combination of these methods. Just to recall some of them: greater number of immediate successors, greater number of successors, smallest upper bound, smallest upper bound divided by the number of successors, greatest processing time divided by the upper bound, smallest lower bound, minimum slack time, minimum number of successors divided by task slack, etc.

2.2.2. Metaheuristics

Garey and Johnson (Garey, 1979) showed the strong NP-completeness of the BPP (thus casting a serious doubt on an exact and efficient algorithm), yet show that the simple 'first fit descending' (FFD) heuristic uses no more (but sometimes not less) than 11/9 of the optimum number of bins. The idea is straightforward: starting with one empty bin, take the items one by one (classified in decreasing size) and for each of them first search the bins so far used for a space large enough to accommodate it. If such a bin can be found, put the item there, if not, request a new bin. Putting the item into the first available bin found yields the 'first fit' (FF) heuristic. Searching for the most filled bin still having enough space for the item yields the 'best fit', a seemingly better heuristic, which can, however, be shown to perform as well (as bad) as the FF, while being slower.

Fitted with acyclic precedence constraints (Sacerdoti, 1977), the BPP becomes the ALBP. A modification of the FFD heuristic (augmenting the size of objects with the

size of all their predecessors) yields a simple heuristic recommended by (Chow, 1990). Given its nature, its performance is similar to the FFD heuristic.

Peterson (Peterson, 1993) developed a *tabu search* (TS) procedure to solve the SALBP. The TS is a meta-strategy for guiding improvements heuristic to overcome local optimality. More detailed descriptions can be found in (Glover, 1997). The solution of the ALBP is adjusted according to *Tabu* restrictions in an attempt to improve the solution to a near-optimal condition.

Sureh and Sahu (Sureh, 1994) developed a *simulated annealing* (SA) (Kirkpatrick, 1983) heuristic for the ALB problem. They considered the problem of stochastic task durations but did not allow parallel stations nor consider multi-products. They only considered single objective problems. Their aim is to minimise the smoothness index introduced in (Moodie, 1965). The smoothness index intends to distribute work among stations as evenly as possible

McMullen and Frazier (McMullen, 1998) presented a SA method to address the ALB. The trade, transfer, compression as well as expansion heuristics are used as local improvements. The initial solution is based upon the 'incremental utilisation' heuristic (Gaither, 1996). The presented approach was used for solving a mixed-model ALBP with stochastic task times and paralleling of tasks within stations.

Minzu (Minzu, 1997) proposed a 'kangaroo' algorithm (a stochastic descent method) to treat the problem of assembly line with a fixed number of stations. The stochastic descent method aims to minimise the maximum work content of the stations, which leads to a well balanced line. The computational tests proved that the 'kangaroo' algorithm supplies good solutions in a small number of iterations.

On the evolutionary algorithm side, to the best of our knowledge, the first attempts was by Falkenauer and Delchambre (Falkenauer, 1992) who used a GGA. The authors generalised their bin packing GGA to obtain a fast algorithm supplying high-quality approximated solutions of the ALBP. One advantage of their method was its ability to handle problems with sparse, even empty precedence constraints, thanks to its bin packing 'ancestor'. Another advantage lies with the fact that the GA is a gradual improvement method, thus giving the user the possibility when the computational resources (time in particular) allow it to extend the search and to continue to improve the currently available solution. Note that the mechanism of complying with precedence constraints proposed in (Falkenauer, 1992) applies equally well to the hybrid GGA of (Falkenauer, 1996), leading to an equally powerful algorithm for SALBP-1.

Anderson and Ferris (Anderson, 1994) showed the effective use of GAs in many combinatorial optimisation problems (COP). The authors used the station-oriented coding: if task *i* is assigned to station *j*, the station number *j* is placed at the *i-th* position in the string. They used the COMSOAL heuristic to initialise the population (Arcus, 1966). They also suggested three types of approaches to deal with infeasible solutions. The first one uses some penalty function to drive the solutions towards

feasibility. A second approach is to force each solution generated to correspond to a feasible one–a repair routine is used to adjust the solution so that it becomes feasible. The third approach can be described as a decoding of the string. The string remains unaltered in the population, but it is decoded using rules which guarantee a feasible assignment. The aim of the authors was only to give some indications for the potential use of this technique in COP rather than to demonstrate the superiority of a GA over traditional methods. The authors also proposed a parallel version of the GA and give some comparisons between the parallel and the serial implementations.

Leu et al. (Leu, 1994) proposed a sequence-oriented GA based method to deal with the SALBP. The authors used a set of heuristics to initialise the population. They also proposed a number of techniques to deal with the feasibility problems during initialisation of the population as well as after the reproduction phase.

Kim et al. (Kim, 1996) developed a GA to solve multiple objective ALBP. They addressed several types of ALBP. They considered the following objectives: (1) minimise number of stations; (2) minimise cycle time; (3) maximise workload smoothness; (4) maximise work relatedness (interrelated tasks are allotted to the same station as much as possible); and (5) a multiple objective with (3) and (4). The authors used a sequence-oriented coding, as well as a repair routine to deal with infeasible solutions. A set of ordering-oriented crossover and mutation operators were used. Their emphasis is placed on seeking a set of diverse Pareto optimal solutions. The multiple objective GA seems to be very promising, in contrast the encoding scheme is not well suited for the grouping problem they deal with.

Sureh (Sureh, 1996) used a GA to solve the single model stochastic version of ALBP. A modified GA, working with two populations (one allowing infeasible solutions), and exchange of specimens at regular intervals, is proposed for handling irregular search spaces, i.e. the unfeasibility problem due to precedence relations. The authors claimed that a population of feasible solutions would lead to a fragmented search space, thus increasing probability of getting trapped in a local minimum. Infeasible solutions can be allowed in the population only if the genetic operators can lead to feasible solutions from infeasible ones. Some solutions are exchanged at regular intervals between the two populations; the exchanged solutions have the same rank of fitness value in their own populations. The results of the experiments indicated that the GA working with two populations can give better results than the GA with only feasible population.

Ponnambalam et al. (Ponnambalam, 1998) used the sequence-oriented representation for a multi-objective GA (MO-GA). They used 14 simple heuristics (Talbot, 1986) to initialise the population and used classical genetic operators to evolve solutions. The method aims too maximise the line *efficiency* as well as the *smoothness* index. During the execution of the MO-GA, a tentative set of Pareto optimal solutions are stored and updated at each generation.

Although different variations of the ALB problem have been explored by researchers (Scholl, 1999), little concern has been given to the physical demands placed on

workers when assigning tasks to stations. Carnahan et al. (Carnahan, 1999) proposed a methodology for the ALB considering both production objectives (cycle time and number of station) as well as worker physical constraints. The method takes into account the physical requirements of tasks. The method is based on the GA and applied to the SALBP-2 where the aim is to minimise the cycle time for a given number of stations. The authors used three search algorithms: (1) a multiple rank heuristic (MRH), (2) a combinatorial GA, and (3) a problem space GA (PSGA). The MRH is a combination of 81 separate heuristics that utilises three search methods (lower bound, upper bound and binary search), three ranking criteria (rank positional weight, number of successors, number of paths), three tasks assignment (forward, backward, bi-directional) and three weighting factors. The sequence-oriented coding combinatorial GA assigns tasks to stations as the classical GA, the main difference is the fact that the number of stations is fixed. Thus, the cycle time constraint is not enforced when assigning tasks to the last station. This ensures that all tasks will be assigned. Tests let the authors to conclude that the PSGA perform better than the MRH and the classical GA.

Lee et al. (Lee, 1999) presented a GA to deal with ALBP with parallel stations. The GA is hybridised with two local search procedures: the trade and transfer heuristics. The trade procedure is used to exchange tasks between two adjacent stations. The transfer procedure has two varieties: the expansion transfer aims to create additional stations if needed, while the compression transfer aims to decrease the number of stations. The method aims to minimise the weighted sum of (1) the total cost of the line and (2) the lateness across all stations. They used the sequence-oriented representation–the tasks are sequentially listed in the order in which they are assigned to the stations. This representation is clearly not grouping-oriented and leads to an inefficient encoding.

The sequence-oriented representation is also used in (Mapfaira, 1999) as an encoding technique. The authors put the accent on the initialisation of population phase. The first step is the creation of the precedence table (tasks with their predecessors). Next, a list of tasks without predecessors is compiled, and then a task without a preceding task is randomly selected from the list and assigned to the first position of the solution string. The assigned task is deleted from the precedence table, the list of tasks without predecessors is updated and another task without preceding is selected and assigned and so on. Solutions are evaluated based on design objectives and are ranked according to their fitness. A one-point crossover is used as reproduction method. The mutation operator works as follows: all tasks to the left side of the mutation point (randomly chosen) are deleted from the chromosome, the initialisation procedure is called then to construct the solution.

Sabuncuoglu et al. (Sabuncuoglu, 2000) proposed a sequence-oriented GA similar to the GA-ordering encoding to deal with ALBP. They used the two-point crossover that the genes in between the cut-points are scrambled according to the order they have in their parent. Such operator ensures the feasibility of the offspring of two feasible parents. In the *scramble* mutation, a random cut-point is selected and the genes after the cut-point are randomly replaced (scrambled), to ensure the feasibility

of the chromosome. The fitness function is composed of two objectives aiming to minimise the number of stations and obtaining the best balanced stations. The well known 'roulette wheel selection' is used. The authors also proposed a method called 'dynamic partitioning' that modifies chromosome structure of GAs to save CPU time. It modifies the chromosome structure by allocating tasks to stations (i.e. freezing certain tasks) that satisfy some criteria, and continues with the remaining unfrozen tasks.

3. Equal piles for the assembly line balancing

The classical ALB approaches tend to group operations under precedence and cycle time constraints. This generally does not yield a desired number of balanced stations. Indeed, even if the line is balanced for a given cycle time, it will never follows such rhythm. There will be (and for ever) failures and uncontrolled behaviours of the line. More, when focus is set on cycle time this typically results in an imbalanced allocation of tasks among stations (Rachamadugu, 1991). Thus, the word *balancing* using when cycle time as constraint is not the appropriate one. SALBP-1 or SALBP-2 methods do not aim to minimise the imbalance between stations. In general, a kind of *local* optimisation is used after global optimisation. This local optimisation yields better allocation of work in the case of SALBP-1 and SALBP-2.

In general, the imperfect balance of stations may be caused by many phenomena like:

1- the stochastic nature of the tasks process times;
2- the station blocking and starving;
3- the machines failures and the operators tiredness.

The symptoms of bad balance impair the performance of the assembly line. Of course, there are tradeoffs associated with any solution to an open problem–our choice of the 'equal piles' concept to address the problems identified above is not exception. As no efficient computational methods leading to exact solutions are known, generally heuristic methods are used to tackle it.

In general, the tasks duration is subject to uncertainty, thus the SALBP-1 are hardly applied to industrial problems. Indeed such methods tend to satisfy a hard constraint: the station process time must not exceed cycle time. Such methods do not care about the number of stations, or generally the space attributed to a given line is fixed. Thus, solutions where the number of stations exceeds the desired number of stations must be discarded. This is why, the accent should be put more on methods dealing with a fixed number of stations.

The SALBP-2 method aims to minimise the cycle time for a fixed number of stations and this may lead to unbalanced line. We propose the equal piles approach to balance assembly line. This method seeks to assign tasks to a fixed number of stations in such a way that the workload of each station is nearly equal.

When dealing with manual multi-product assembly line, in a batch oriented approach, most of the time the designer tends to reuse and rebalance the same line for the different batches. As the allocated space to the AL (number of stations, heavy machines, etc.) is generally fixed, the new balance have to deal with these constraints. This also may occur when the designer has to reuse existing stations to assemble new products. In both cases the aim is most of the time to equalise stations workloads, which is known as the equal piles for assembly line problem (EPALP).

3.1. Motivation and inspiration from nature

The simple equal piles problem (which does not take precedence constraints into account) was first defined by Jones and Betramo (Jones, 1991) as follow. Given a set of N objects of various sizes (one dimension), distribute the objects into *K* ‘piles’ in such a way that the ‘heights’ of the piles are as equal as possible. To solve this problem, Falkenauer presented a GGA combined with the ‘first fit descending’ heuristic (Falkenauer, 1995). Having initialised the required number of groups at random, the remaining items are sorted in decreasing size, and successively added to the ‘smallest pile’.

The method presented in this book was inspired from nature, by the observation of the boundary stones along the motorway. Indeed, the distance between two cities is fixed by the number of boundaries (which is equivalent to the number of kilometres). In the case of flat motorway and fixed speed, the time to cover the distance between two adjacent boundaries is more-or-less the same (well balanced). It is known that in order to reduce the time to cover the distance between two cities (a distance being fixed) we must increase the speed. More, aiming to have an average speed equal to the theoretical one, the safely way is (1) to get back missed time between two boundaries by speeding up between others, or (2) to lose time earned between two boundaries by slowing down the speed between others.

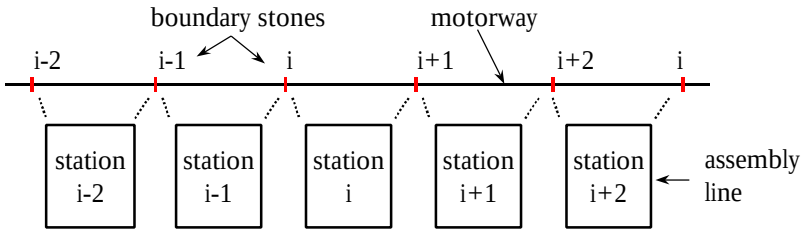

Figure 6.1. Correspondence between an assembly line and a motorway.

In the case of assembly line, the space is fixed by the designer. The speed of the line depends on the desired throughput of the line and the process time of the tasks. In this case the distance between two boundaries is equivalent to the process time of a station (Figure 6.1). Thus, the product has to cross the line in a duration fixed by the desired cycle time and the number of stations. More, aiming to keep the real throughput close as possible to the desired one, a safe way is (1) to get back missed

process time on a given station by speeding up others, or (2) to lose time earned on a given station by slowing down the speed of others. The hard constraint is not anymore the cycle time but the number of stations.

The proposed correspondence between an assembly line with a fixed number of stations and a flat motorway was used to implement the EPAL algorithm to deal with the balancing of the workload among stations.

3.2. Definition of the equal piles for assembly line problem

The method helps to solve the problem of stations space and load balancing among existing stations and warrants to obtain the desired number of stations. So, the problem is to group tasks into a fixed number of stations so that:

1. The precedence relations between tasks are preserved, i.e. no task can be done unless all its preceding work elements have been completed;
2. The process time on the different stations are nearly equal.

3.3. Input data

The proposed approach needs the following input data as illustrated on Figure 6.2:

- the desired number of stations;
- the duration of each operation;
- the assembly plan (precedence graph) of the product;
- the user's preference constraints (associative and dissociative).

Figure 6.2. Data flow for equal piles assembly line.

The two first input data are used to estimate the desired cycle time of the assembly line. The other ones allow us to deal with the user's needs (sequence of operations, separate some operations and group some others, etc.).

3.4. Customising the GGA to the EPAL problem

The proposed method (heuristic) is based on the grouping genetic algorithm (GGA). The main features of GGA can be found in Chapters 4 and 5. The structure of the proposed EPAL-GGA is as follows.

Generate an initial population with an individual construction algorithm (ICA);
repeat
Select parents;
Recombine best parents from the population;
Mutate children;
Reconstruct individuals using the ICA;
Replace individuals of the population by children;
until *a satisfactory solution has been found.*

In the next section the individual construction algorithm (ICA) is introduced. The purpose of the ICA (EPAL) heuristic is to allocate tasks to a fixed number of stations. The EPAL approach allows to deal with the precedence constraints between tasks. The EPAL approach is embedded in the GGA and is used to construct the first population as well as to re-construct the solutions at each generation. The essential and distinct concepts adopted by the method will be described below, along with step-by-step execution procedure and an illustrative example.

3.4.1. Boundary stones algorithm

As said above, in the equal piles problem, the hard constraint is the fixed number of stations (piles). The approach we propose to solve the problem is based on what we call the boundary-stones. These boundaries will be used as seeds to fill stations. The algorithm follows the steps described below.

Step 1:
It begins by detecting if the precedence graph is cyclic (Sedgewick, 1984). This step allows to check the validity of the proposed precedence graph of the product.

Step 2:
It orders the operations using the labels defined below. The labels of tasks depend on the number of their predecessors and successors. A first formula is given by

$$label1_i = nbpreds_i - nbsuccs_i \quad \text{(Equation 1)}$$

where $label1_i$ is the ordering criteria of operation i (it heavily depends on the precedence graph), $nbpreds_i$ is the total number of predecessors of operation i, and $nbsuccs_i$ is its total number of successors. This formulation of labels does not take into account the operation durations.

For complex graphs (presenting several sprays), formula (1) falls in a trap (yielding a poor balancing). The more sprays the graph contains the more its ordering becomes difficult (for a graph having a diagonal adjacency matrix the labelling is very easy since the graph is 'dense' and uniform). Formula (1) was completed to avoid this pitfall, giving

$$label_i = nbpreds_i - nbsuccs_i + follows_i \qquad \text{(Equation 2)}$$

where $follows_i$ is the maximal $label1_i$ value of the direct successors of operation i in the precedence graph.

Table 1 gives the application of formulas (1) and (2) to the precedence graph illustrated in Figure 6.3.

For instance, operation 1 has 0 predecessors and 4 successors, thus, $label_1(1)=0-4=-4$. The direct successors of 1 are {3, 4} and their respective labels are {-1, 0}, thus the maximal value is (Follow(1)=0). The label of operation 1 according to formula (2) is -4.

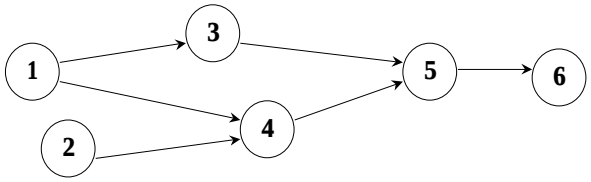

Figure 6.3. Example of precedence graph.

Operation	Duration	nbpreds	nbsuccs	label1	Follows	label
1	4	0	4	-4	0	-4
2	2	0	3	-3	0	-3
3	3	1	2	-1	3	2
4	3	2	2	0	3	3
5	1	4	1	3	5	8
6	5	5	0	5	6[1]	11

Table 6.1. Example of applications of equations (1) and (2).

This method permits to order operations, finding the probable first and last operations on the product and permits to choose the possible seeds of stations (the boundary stones).

Step 3:
Boundary stones (or station seeds) are chosen using the sequence obtained at the second step. The number of stones is equal to the number of stations. This step allows us to find seeds of piles (or stations). Suppose that the aim is to balance an assembly line with three stations (so the number of stones is three). The boundary stones are determined to group operations in three clusters corresponding to the three stations. In this example, the first operation in the precedence graph of the

[1] There are no successors for the given operation, and it is the last operation in the precedence graph, so the total number of operations is taken as the value of the label.

product is *1* (it has no predecessor and gets the minimal *label*). The last operation is *6* (having no successor and corresponding to the maximal *label*).

According to their labels {-4, -3, 2, 3, 8, 11}, operations are ordered as follows {1, 2, 3, 4, 5, 6} (refer to Table 6.1). The first boundary stone is the label corresponding to the first operation:

$$stone_1 = \min(label) \qquad \text{(Equation 3)}$$

The boundary stone *i* is defined as:

$$stone_{i+1} = stone_i + gap \qquad \text{(Equation 4)}$$

where

$$gap = \frac{\max_i(label_i) - \min_i(label_i)}{N} \qquad \text{(Equation 5)}$$

In this example, gap=5 and the boundary stones are {-4, 1, 6}.

Step 4:
Once the boundary stones have been fixed, the labels (and consequently operations) are grouped into as many clusters as stations. The seed (to which corresponds the first operation) of cluster *i*, $seed_i$ will be a *label* close to $stone_i$; for the first station, we fix $seed_1 = label_1$. To this $seed_i$ will also correspond an operation which will be the seed of station *i*. Note that there can be several possible seeds (operations) for each cluster, which adds randomness to the procedure. Once the seeds have been selected, each cluster *i* is completed by adding *label* (s) to it in increasing order, so that:

$$\forall cluster_i, \forall j \in [1, n_{op}], seed_i \le label_j < seed_{i+1}, \qquad \text{(Equation 6)}$$

where n_{op} is the total number of operations. The clustering fixes the possible insertion positions (stations) of the remaining unassigned operations. Operations of $cluster_i$ (the one corresponding to the $seed_i$ and the operations in the last cluster excepted) may be assigned to station *i* or *i*+1.

For example, suppose the chosen cluster seeds are {-4, 2, 8}. The corresponding label clusters are {-4, -3}, {2, 3} and {8, 11}. So operation 1 will be assigned to station 1, operation 3 to station 2 and operation 5 to station 3. Among the remaining operations, forming three clusters {2}, {4}, {6}, operation 2 may be assigned to stations 1 or 2, operation 4 to station 2 or 3, and operation 6 to station 3. Figure 6.4 illustrates our words. The operations already assigned are the station seeds. The arrows starting from the clusters (cl1, cl2, cl3) point to the station which the remaining operations can be assigned to.

Step 5:
Once the clustering has been done the algorithm assigns the remaining operations to stations according to the rules exposed at step 4, taking into account the precedence

constraints (see section 3.4.2) and the user's preferences. The operations are randomly extracted from the clusters.

Due to the precedence constraints of the product, most of the time some station loads will exceed the desired cycle time (the maximum stations process time). A local improvement phase attempts to equalise again those loads, by moving operations along the line or exchanging operations between stations (see section 3.4.3).

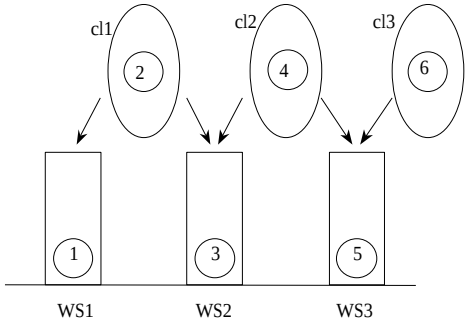

Figure 6.4. Operation clustering and assignment.

Steps 4 and 5 are applied each time the GA is about to construct a new solution, e.g. at the population initialisation, or completing an existing one, e.g. after a crossover or during the mutation.

3.4.2. Dealing with precedence constraints

The EPAL algorithm must yield a solution which respects all precedence constraints of the product. Indeed, if these constraints are violated for a given order of stations along the line the product being assembled has to move against the sense of the line conveyor, thus visiting at least one station several times.

Each time we are about to assign an operation to a station:

1. We look for the 'boundary-stone' corresponding to the given operation,
2. Let 'X' be the station where to insert the operation. We check if the stations corresponding to the already assigned successors of the given operation precede 'X'. If it is the case, we look for another station where to insert the operation, avoiding the violation of precedence constraints.

3.4.3. Heuristics

Two heuristics are used to improve the solutions obtained by the boundary stones algorithm: the *simple wheel* and the *multiple wheels*. Both of them will be executed on a solution until no improvement is obtained or a maximum number of trials is reached.

Simple wheel:
This heuristic moves a set of operations along the line. The move will always be accepted if the destination station process time added to the process time of the moved operations does not exceed the cycle time. If it exceeds, the move is accepted with some probability. Firstly, the heuristic tries to move a set of operations from the first station to the second one. Then it tries to move a set from the new second station to the third and so on until the last station is reached. Next, it begins with moves from the last station to the last but one and so on until the first station is reached (see Figure 6.5). This leads to move operations along the line to reduce the imbalance between stations (precedence constraints are always checked).

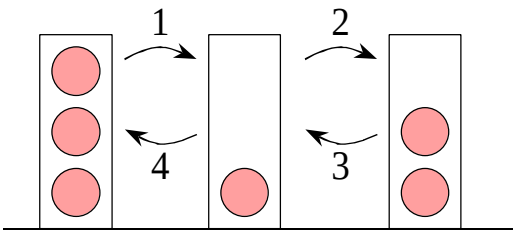

Figure 6.5. Simple wheel heuristic.

Multiple wheel:
The second idea is to exchange operations between stations. Two adjacent stations are taken at each time (refer to Figure 6.6). All possible exchanges (which do not violate precedence constraints) are executed. The first exchange is made between the first and second station, the second one is performed between the second and third station and so on.

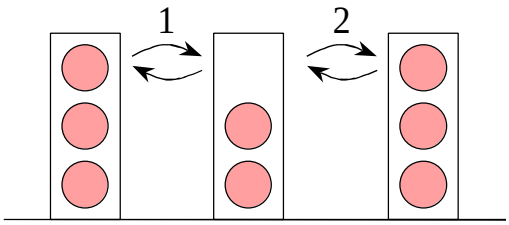

Figure 6.6. Multiple wheel heuristic.

The first heuristic gives the best results, the second is used only if the algorithm is stuck in local optima and fails to improve the solution.

3.4.4. Cost function

The objective is to equalise stations loads, under the constraint of a fixed number of stations. So the first cost function which comes to mind is simply the sum of the differences between the stations operating times and the desired cycle time. Such a cost function lacks any capacity of guiding the algorithm in its search. Indeed, a

solution where the duration of half the number of stations exceeds the cycle time and the duration of the second half is below the cycle time seems to be a good one as illustrated in Figure 6.7.

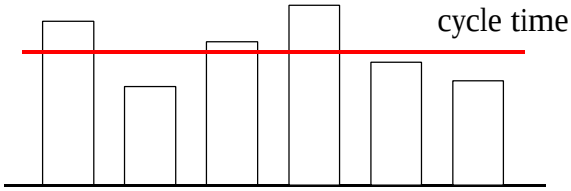

Figure 6.7. Badly balanced line with 'negative' imbalance equal to 'positive' one.

We propose the following cost function (balance index), which has to be minimised:

$$f_{EP} = \sum_{i=1..N} (fill_i - cycletime)^2 \quad \text{(Equation 7)}$$

where N is the number of stations, $fill_i$ the sum of working times on station i, $time_i$ is the process time of task i, and $cycletime$ the desired cycle time, defined as:

$$cycletime = \frac{\sum_{i=1}^{nop} time_i}{N} \quad \text{(Equation 8)}$$

3.5. Experimental results

In order to assess the merit of the proposed algorithm, many tests were done on benchmarks of the suite proposed by (Scholl, 1999). The proposed metaheuristic was tested on a set of instances (BUXEY, GUNTHER, HAHN, KILBRIDGE, LUTZ, WARNEKEE, WEE-MAG) taken from the benchmarks proposed by (Scholl, 1999). More details about these benchmarks can be on the net[2].

The EPAL-GGA used in the experiments is a steady-state group-based GA using a population of *36* individuals. Seven instances were used to test the method. The EPAL was applied five times on each instance. For each instance the results are given in tabular and graphical forms. The first table gives the maximal and the minimal process time of each solution. The second table summarises the balancing of the solution.

Table 6.2 summarises the results corresponding to 'BUXEY' instance. BUXEY instance is composed of 29 operations and the total process time is 324. CT represents the theoretical cycle time corresponding to a given number of stations. For a given number of stations N, CT is the sum of process time of all tasks divided

[2] The benchmark suite can be accessed via the Web at
http://www.bwl.tu-darmstadt.de/bwl3/forsch/projekte/alb/index.htm

by N. Min (respectively Max) represents the minimal (respectively maximal) process time of stations of a given solution.

N	CT	Min 1	Max 1	Min 2	Max 2	Min 3	Max 3	Min 4	Max 4	Min 5	Max 5
7	46.2857	45	48	45	48	44	47	44	47	45	48
8	40.5000	40	41	40	41	40	41	40	41	40	41
9	36.0000	32	38	32	38	32	38	32	38	32	38
10	32.4000	27	35	30	34	26	35	27	35	30	35
11	29.4545	20	34	20	33	20	34	20	34	20	33
12	27.0000	23	29	24	29	24	29	24	29	25	28
13	24.9231	20	28	20	28	20	28	20	28	20	28
14	23.1429	20	26	15	26	14	27	20	26	20	26

Table 6.2. BUXEY's minimal/maximal workload of stations.

N	Balance 1	Balance 2	Balance 3	Balance 4	Balance 5	Avg Run Time	Std Deviation
7	0.0503	0.0503	0.0589	0.0589	0.0503	29.16	14.774
8	0.0349	0.0349	0.0349	0.0349	0.0349	4.0600	0.171
9	0.1470	0.1470	0.1470	0.1521	0.1470	2.7400	0.475
10	0.2147	0.1525	0.2438	0.2191	0.1757	26.5400	3.087
11	0.3970	0.4168	0.4277	0.4462	0.4140	33.1000	9.507
12	0.2029	0.1737	0.1889	0.1737	0.1283	38.9400	8.911
13	0.3784	0.3654	0.3654	0.3426	0.3698	46.7400	13.073
14	0.3339	0.4567	0.5034	0.3449	0.3225	59.7600	10.598

Table 6.3. BUXEY's balancing.

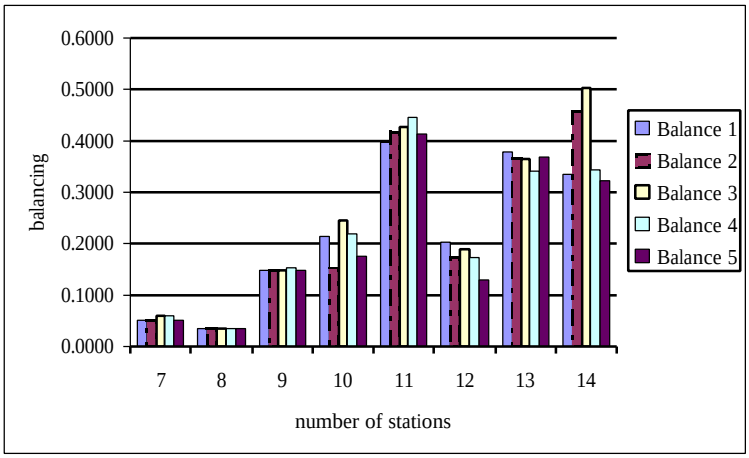

Figure 6.8. The balancing versus the number of stations (BUXEY).

Table 6.3 gives the balancing corresponding to the 5 runs (the balance index was obtained using Equation 7 (see section 3.4.4)) of the best solutions. The last two columns represent respectively the average run time (on a PENTIUM II 333 MHz) and the standard deviations of the values. Figure 6.8 represents the balancing versus the number of stations. The obtained balancing is in general less than 1 (1 corresponds to a highly imbalance assembly line). Results also show that the

balancing decreases (non balanced stations) with the number of stations. Indeed, bigger is the number of stations smaller is the cycle time and smaller is the number of tasks by stations is smaller. Thus, the balancing becomes complicated. This explains why balancing values corresponding to 14 stations (Figure 6.8) are quite big.

Appendix 4 gives the results obtained respectively for GUNTHER, HAHN, KILBRIDGE, LUTZ, WARNEKEE, WEE-MAG.

The results show that the standard deviations are high, but the average run time still reasonable so we can conclude that the algorithm behaves uniformly. The results obtained on these benchmarks show also the powerful of the proposed method. Indeed, the approach allows to balance the workload of a given AL for a fixed number of stations. Chapter 10 presents results of EPAL on an industrial case study.

4. Extension to multi-product assembly line

In formal terms we define the multi-product assembly line problem with a fixed number of stations (MPALPF) as the following decision problem.

Given a directed acyclic graph G=(T, P) (the nodes of T representing the tasks and the arc of P representing the precedence constraints) with a distribution *frequency* and an *average* duration of each task L_i (task length), assigned to each node T_i, and a constant N (number of station), can the nodes of T be partitioned into the N subsets S_j (the j-th station tasks) in such a way that: there exists an ordering of the subsets such that whenever two nodes in distinct subsets are joined by an arrow in G, the arrow goes from a higher ordered (earlier one) to a lower ordered (later one)?

4.1. Multiple objective problem

The multi-product assembly line balancing with a fixed number of stations may have two conflicting objectives:

1. equalise the average station load (EPAL);
2. minimise the difference between product-variants workload on each station.

This two objectives are conflicting in the sense that: the first objective may lead to line balanced on average (by minimising the average idle time), whereas the second may drift from the first one, and leads to an even workload for the different variants on a given station. The results obtained using one of these objectives are generally different. To illustrate this, suppose we have M=4 operations and N=2 stations and two variants. Suppose the following variant operation's duration ({0, 5},{4, 4},{5, 0},{6, 6}), and the two solutions given in (Figure 6.9).

The first solution presents a good balancing of the stations on average, and a great difference between variants process times. On the other hand the second one

presents a bad average balancing among the stations but no difference between the two variants process time. The choice among these two solutions becomes hard, and depends on the preferences of the designer. The method we propose tends to deal with this problem by taking into account the user's preferences. It helps to balance the workload among stations and warrants to obtain the desired number of stations. Thus, the problem is to group tasks into a fixed number of stations so that:

1. the precedence relations are preserved, i.e. no task can be done unless all its preceding work elements have been completed;
2. the process time on the different station are nearly equal on average;
3. the variants process times on each station are as equal as possible.

	op	pt_ws	pt_var1	pt_var2
Ws1	1,2	6.5	4	9
Ws2	3,4	8.5	11	6

	op	pt_ws	pt_var1	pt_var2
Ws1	1,3	5	5	10
Ws2	2,4	10	5	10

WsX: station X,
Op: list of operations,
pt_ws : station process time,
pt_varY: process time on variant Y,

Figure 6.9. Two grouping and their corresponding process times.

4.2. Overall architecture

The line balance efficiency is impacted by the average process time of each station along the line (the average idle time) as well as the imbalance between variants process times on each station. Normally, the fewer the number of stations in the line and the less the idle time, the more efficient the line. Thus we have to decide which task will be performed on which station. The proposed algorithm is based on the EPAL approach and the multiple objective grouping genetic algorithm (MO-GGA) (see Chapter 5). The main steps of the algorithm that serve to generate possible solutions to the problem are summarised below. This method will be applied each time we have to construct or complete the construction (after a crossover for example) of an individual in the GGA. The GGA steps are the following.

1) Create a population of individuals using EPAL method (see section 3).
2) Use the decision-aid method PROMETHEE II to order individuals in the population.
3) Recombine (mate) best individuals (parents) to produce children.
4) Mutate children.
5) Use PROMETHEE II to order the new population.
6) Replace the worst individuals of the population by the new children.
7) ***If** a satisfactory solution is found stop. **Else** go to 3).*

4.2.1. Input data

The EPAL for multi-product assembly line algorithm needs the following input data as illustrated on Figure 6.10:

- desired number of stations;
- duration of each operation;
- list of variant products;
- precedence graph of the product family;
- users preferences.

The two first input data let us estimate the average cycle time of the assembly line. The third one deals with the mixed-production problem. On the other hand, the two lasts allow us to deal with the user's needs.

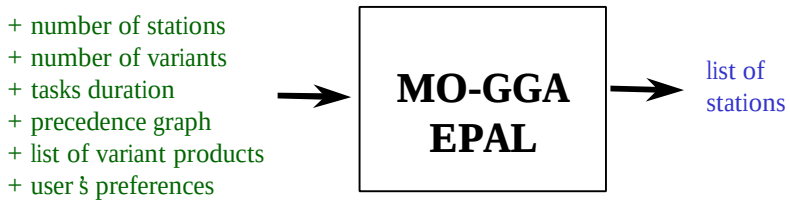

Figure 6.10. Data flow for equal piles assembly line.

In order to assign operations to stations, we use the ICA heuristic described for the EPAL problem.

4.2.2. Performance of the solutions

We here present the two conflicting objectives used to design MPALs:

Process time divergence among products variants:

The process time of station w on variant i is given by (Equation 9). This process time is the sum of the process time of the tasks j of station w on variant i ($Ptime(i, j)$ is the process time of task j on variant i). The number of tasks of each station w is set to M_w.

$$Variant\_Ptime(i,w) = \sum_{j=1..M_w} Ptime(i, j) \qquad \text{(Equation 9)}$$

The process time of operation j is the ratio between the sum of process time of task j and the different variants on the number of variants.

$$Operation\_Ptime(j) = \frac{\sum_{i=1..Nb\_Variant} Ptime(i, j)}{Nb\_Variant} \quad \text{(Equation 10)}$$

The process time of station *w* is the sum of the process time of the tasks that belong to station *w*.

$$Ws\_Ptime(w) = \sum_{i=1..N} (Operation\_Ptime(i)) \quad \text{(Equation 11)}$$

Equation 12 presents the standard deviation of the process time on station *w*.

$$Std\_div(w)^2 = \frac{\sum_{i=1..Nb\_Variant} (Ws\_Ptime(w) - Variant\_Ptime(i, w))^2}{Nb\_Variant} \quad \text{(Equation 12)}$$

The line standard deviation of the process time is the ratio between the sum of standard deviation of the different stations and the number of stations.

$$Line\_Std\_Div = \frac{\sum_{i=1..N} (Std\_div(i))}{N} \quad \text{(Equation 13)}$$

Imbalance of the line:

The imbalance of the assembly line the ratio of the sum of the square of the difference between the desired cycle time and the process time of stations and the cycle time.

$$Line\_\mathrm{Im}balance = \frac{\sum_{i=1..N} (CT - Ws\_Ptime(i))^2}{CT} \quad \text{(Equation 14)}$$

The ideal cycle time *CT* is the ratio of the sum of the process time of tasks and the number of tasks *N*.

$$CT = \frac{\sum_{i=1..N} Operation\_Ptime(i)}{N} \quad \text{(Equation 15)}$$

where :

N	Number of stations.
M_w	Number of operations of station *w*.
CT	Desired assembly line cycle time.

Nb_Variant	Number of variants.
Line_Std_Div	The standard deviation of assembly line.
Line_Imbalance	The imbalance among station.
Std_Div(*i*)	The standard deviation of station *i.*
Variant_Ptime(*i*,*w*)	Process time of station *w* on variant *i.*
Ws_Ptime(*w*)	Station process time.
Ptime(*i, j*)	Process time of operation *i* on variant j.
Operation_Ptime(*i*)	Process time for operation *i.*

The cost function is the following:

$$Cost\ Function = PROMETHEE\ (Line\_Std\_div, Line\_Misbalance)$$ [3]

It is easy to show that in the case of multi-product assembly line, the better each station is used, the fewer stations one needs to group all tasks in. Or, conversely, a bad use of station capacity leads to the necessity of supplementary stations, in order to contain the tasks not packed into the wasted space. On the other hand, a large imbalance of the workload among different variants has to be avoided. Indeed, even when the load of the stations is balanced on average, the production can easily be faced with work overflow or starvation on individual stations. The work imbalance due to the variants should thus be distributed among the stations. In Chapter 8 are summarised results obtained in the case of multi-product assembly line.

5. Conclusions and further works

In this chapter, we presented a new method to treat 'equal piles for assembly line' problem. The method is based on the GGA. This bin filling philosophy is very different from classical line balancing problem one. Special heuristics where developed to tackle the problem: the 'boundary stones' and 'wheel' heuristics. One of the main aims is the re-balancing of an existing AL, arising after a change in the product design during the operation phase, or between two different production batches. The designer here may wish to keep the former number of stations.

The method was also enhanced to deal with MPAL. The EPAL method embedded in the multiple objective grouping genetic algorithm (MO-GGA) was presented. The aim in this case is to balance the average process time of each station and the to reduce the difference between the different variant. In Chapter 8, the balance for operation concept will be introduced. The aim of this concept is to have the operation phase interacted with the design phase in the case of a mixed production.

The same kernel (EPAL) algorithm which was introduced in this chapter to deal with manual assembly line will be further applied to design hybrid assembly line (see Chapter 7).

---

[3] Where PROMETHEE(x1, x2, ...) means we use the multi-criteria decision-aid method PROMETHEE II, and the objectives are x1, x2, etc.

Tests on the academic case studies presented in this chapter show the power of the developed method. Since EPAL solutions depend on the so-called 'boundary-stones', further research will be undertaken to understand how to ideally choose this stones. The GGA cost function also will be improved to help the GGA in its search for good solutions. For graphs presenting several sprays the operations ordering with the labels is not always pertinent. Further research must be undertaken to enhance the ability of the algorithm to deal with such graphs.

6. References

(Agnetis, 1997) Agnetis A. and Arbib C., 'Concurrent operations assignment and sequencing of particular assembly problems in flow lines', Annals of Operations Research, Vol. 69, pp. 1-31, 1997.

(Anderson, 1994) Anderson E.J. and Ferris M.C., 'Genetic algorithms for combinatorial optimisation: the assembly line balancing problem', ORSA Journal on Computing, Vol. 6, pp. 161-173, 1994.

(Arcus, 1966) Arcus A.L., 'COMSOAL: A computer method of sequencing operations for assembly lines', Int. J. of Prod. Res., Vol. 4, pp. 259-277, 1966.

(Baybars, 1986) Baybars I., 'A survey of exact algorithms for the simple assembly line balancing', Management Science, Vol. 32, pp. 909-932, 1986.

(Boctor, 1995) Boctor F.F., 'A multiple-rule heuristic for assembly line balancing', J. of Oper. Res. Society, Vol. 46(1), pp. 62-69, 1995.

(Bock, 1997) Bock S. and Rosenberg O., 'A new distributed fault-tolerant algorithm for the simple assembly line balancing problem 1', Technical report TR-RSFB-97-040, University of Paderborn, 1997.

(Bowman, 1960) Bowman E.H., 'Assembly line balancing by linear programming', Oper. Res., Vol. 8(3), pp. 385-389, 1960.

(Carnahan, 1999) Carnahan B.J., Redfen M.S., Norman B.A., 'Incorporating physical demand criteria into assembly line balancing', Pittsburgh University, Department of Industrial Engineering, Internal Report TR99-8, 1999.

(Chow, 1990) Chow W.-M., 'Assembly line design - methodology and applications', Marcel Dekker, New York, 1990.

(Falkenauer, 1996) Falkenauer E., 'A hybrid grouping genetic algorithm for bin packing', Journal of Heuristics, Vol. 2(1), pp. 5-30, 1996.

(Falkenauer, 1995) Falkenauer E., 'Solving equal piles with a grouping genetic algorithm', Eshelman L.J. (Ed.), Proceedings of the Sixth International Conference on Genetic Algorithms (ICGA95), Morgan Kaufmann Publishers, San Francisco, pp. 492-497, 1995.

(Falkenauer, 1992) Falkenauer E. and Delchambre A., 'A genetic algorithm for bin packing and line balancing', Proceedings of the IEEE International Conference on Robotics and Automation (RA92), IEEE Computer Society Press, pp. 1186-1192, 1992.

(Gaither, 1996) Gaither N., 'Production and operations management', Belmon, CA: Duxbury, 1996.

(Garey, 1979) Garey M. R. and Johnson D.S., 'Computers and intractability - a guide to the theory of NP-completeness', Freeman, San Francisco USA, 1979.

(Ghosh, 1989) Ghosh S. and Gagnon R.J., 'A comprehensive literature review and analysis of the design, balancing and scheduling of assembly systems', Int. J. of Prod. Res., Vol. 27, pp. 637-670, 1989.

(Glover, 1997) Glover F. and Laguna M., 'Tabu search', Kluwer Academic Publishers, Boston, 1997.

(Gutjahr, 1964) Gutjahr A.L. and Nemhauser N.K., 'An algorithm for the line balancing problem', Management Science, Vol. 11(2), pp. 308-315, 1964.

(Helgeson, 1961) Helgeson W.B. and Birnie D.P., 'Assembly line balancing using the ranked positional weight technique', J. Ind. Engng, Vol. 12(6), pp. 394-398, 1961.

(Hoffmann, 1963) Hoffmann T.R., 'Assembly line balancing with precedence constraints', Management Science, Vol. 9, p. 551-562, 1963.

(Hoffmann, 1992) Hoffmann T.R., 'Eureka: A hybrid system for assembly line balancing', Management Science, Vol. 38, pp. 39-47, 1992.

(Jackson, 1956) Jackson J.R., 'A computing procedure for the line balancing problem', Management Science, Vol. 2, pp. 261-271, 1956.

(Jones, 1991) Jones D.R. and Beltramo M.A., 'Solving partitioning problems with genetic algorithms', Proceedings of the 4th International Conference on Genetic Algorithms, Morgan Kaufmann, pp. 442-449, 1991.

(Johnson, 1988) Johnson R.V., 'Optimally balancing large assembly lines with "FABLE"', Management Science, Vol. 34, pp. 240-253, 1988.

(Kilbridge, 1961) Kilbridge M.D. and Wester L., 'A heuristic method for assembly line balancing', Journal of Industrial Engineering, Vol. 12, pp. 292-298, 1961.

(Kirkpatrick, 1983) Kirkpatrick S., Gelatt C.D. Jr. and Vecchi M.P., 'Optimisation by simulated annealing', Science, Vol. 220, pp. 671-680, 1983.

(Kim, 1996) Kim Y.K., Kim Y.J. and Kim Y., 'Genetic algorithms for assembly line balancing with various objectives', Computers & Industrial Engineering, Vol. 30(3), pp. 397-409, 1996.

(Lapierre, 1999) Lapierre S.D. and Ruiz A., 'Equilibrer une chaîne d'assemblage avec Microsoft ACCESS97', Proceedings of 3rd International Industrial Engineering Conference, Presses Internationales Polytechnique, Montreal Canada, pp. 357-364, 1999.

(Lee, 1999) Lee C.Y., Gen M. and Tsujimura Y., 'Multicriteria assembly line balancing problem with parallel workstations using hybrid GAs', Proceedings of EDA'99, pp. 115-122, 1999.

(Leu, 1994) Leu Y.Y., Matheson L.A. and Rees L.P., 'Assembly line balancing using genetic algorithms with heuristic-generated initial population and multiple evaluation criteria', Decision Sciences, Vol. 25(4), pp. 581-606, 1994.

(Mapfaira, 1999) Mapfaira H. and Byrne M., 'A genetic algorithms approach for assembly line balancing', Proceedings of ICED'99, Vol. 2, Munich, Germany, pp. 957-960, 1999.

(Martello, 1990) Martello S. and Toth P., 'Bin-packing problem', Chapter 8 in Knapsack Problems, Algorithms and Computer Implementations, John Wiley & Sons Ltd., England, pp. 221-245, 1990.

(McMullen, 1998) McMullen P.R. and Frazier G.V., 'Using simulated annealing to solve a multiobjective assembly line balancing problem with parallel stations', International Journal of Production Research, Vol. 36, pp. 2717-2741, 1998.

(Minzu, 1997) Minzu V. and Henrioud J-M., 'Assignment stochastic algorithm in multi-product assembly lines', Proceedings of ISATP'97, pp. 109-114, 1997.

(Moodie, 1965) Moodie C.L. and Young H.H., 'A heuristic method of assembly line balancing for assumptions of constant or variable work element times', J. Ind. Eng., Vol. 16(1), pp. 23-29, 1965.

(Praça, 1999) Praça I.C. and Ramos C., 'Multi-agent simulation for balancing of assembly lines', Proceedings of ISATP'99, pp. 459-464, 1999.

(Patterson, 1975) Patterson J.H. and Albracht J.J., 'Assembly line balancing: zero-one programming with fibonnaci search', Oper. Res., Vol. 23, pp. 166-172, 1975.

(Peterson, 1993) Peterson C., 'A tabu search procedure for the simple assembly line balancing problem', Proceedings of the Decision Science Institute conference, Washington, DC, pp. 1502-1504, 1993.

(Peng, 1991) Peng Y., 'The algorithms for the assembly line balancing problem', Internal Report, CRIF Industrial Automation, Belgium, 1991.

(Ponnambalam, 1998) Ponnambalam S.G., Aravindan P. and Mogileeswar Naidu G., 'Assembly line balancing using multi-objective genetic algorithm', Proceedings of CARS&FOF'98, Coimbatore, India, pp. 222-230, 1998.

(Rachamadugu, 1991) Rachamadugu R. and Talbot B., 'Improving the equality of workload assignments in assembly lines', Int. J. Prod. Res., Vol. 29(3), pp. 619-633, 1991.

(Sacerdoti, 1977) Sacerdoti E.D., 'A structure for plans and behavior', Stanford Research Institute, Elsevier North-Holland Inc, 1977.

(Salveson, 1955) Salveson M.E. , 'The assembly line balancing problem', J. Ind. Engng., Vol. 6(3), pp. 18-25, 1955.

(Scholl, 1999) Scholl A., 'Balancing and sequencing of assembly lines', Heidelberg Physica, 1999.

(Scholl, 1997) Scholl A. and Klein R., SALOME: A bidirectional branch and bound procedure for assembly line balancing', INFORMS Journal on Computing, Vol. 9, pp. 319-334, 1997.

(Sedgewick, 1984) Sedgewick R., 'Algorithms', Addisson-Wessley Publishing, 1984.

(Sabuncuoglu, 2000) Sabuncuoglu I., Erel E. and Tanyer M., 'Assembly line balancing using genetic algorithms', Journal of Intelligent Manufacturing, Vol. 11, pp. 295-310, 2000.

(Shtub, 1990) Shtub A. and Dar El E.M., 'An assembly chart oriented assembly line balancing approach', Int. J. of Prod. Res., Vol. 28(6), pp. 1137-1151, 1990.

(Süer, 1998) Süer G.A., 'Designing parallel assembly lines', Computers Ind. Eng., Vol. 35 (3-4), pp. 467-470, 1998.

(Sureh, 1994) Sureh G. and Sahu S., Stochastic assembly line balancing using simulated annealing', Int. J. Prod. Res., Vol. 32(8), pp. 1801-1810, 1994.

(Sureh, 1996) Sureh G., Vinod V.V. and Sahu S., 'A genetic algorithm for assembly line balancing', Production Planning and Control, Vol. 7, pp. 38-46, 1996.

(Talbot, 1986) Talbot F.B., Patterson J.H., Gehrlein W.V., 'A comparative evaluation of heuristic line balancing techniques', Management Science, Vol. 32, pp. 430-454, 1986.

(Wee, 1982) Wee T.S. and Magazine M.J., 'Assembly line balancing as generalized bin packing', Operations Research Letters, Vol. 1, pp. 56-58, 1982.