J Intell Manuf (2008) 19:49–69
DOI 10.1007/s10845-007-0045-5

A review of the current applications of genetic algorithms in assembly line balancing

Seren Ozmehmet Tasan · Semra Tunali

Received: April 2006 / Accepted: December 2006 / Published online: July 2007

Abstract Most of the problems involving the design and plan of manufacturing systems are combinatorial and NP-hard. A well-known manufacturing optimization problem is the assembly line balancing problem (ALBP). Due to the complexity of the problem, in recent years, a growing number of researchers have employed genetic algorithms. In this article, a survey has been conducted from the recent published literature on assembly line balancing including genetic algorithms. In particular, we have summarized the main specifications of the problems studied, the genetic algorithms suggested and the objective functions used in evaluating the performance of the genetic algorithms. Moreover, future research directions have been identified and are suggested.

Keywords Assembly line balancing · Genetic algorithms · Chromosome representation · Fitness evaluation · Genetic operators

Introduction

Today's highly competitive, fast pace business environment makes it an absolute requirement on manufacturers to continuously and effectively optimize the design of manufacturing systems in the shortest possible time. These conditions require a responsive manufacturing system that could be rapidly designed, able to convert quickly to the production of new product models, able to adjust capacity quickly, able to integrate process technology and able to produce an increased variety of products in unpredictable quantities. In this business environment, the design of such manufacturing systems, which involves the design of products, processes and the plant layout before physical construction, becomes more and more important. Particularly, the design of an efficient assembly line has a considerable industrial importance (Baudin, 2002). A well-known assembly design problem is the assembly line balancing problem (ALBP). ALBP deals with the allocation of the tasks among workstations so that the precedence relations are not violated and a given objective function is optimized.

ALBP falls into the NP-hard class of combinatorial optimization problems (Karp, 1972). The complexity of the ALBP renders optimum seeking methods impractical for instances of more than a few tasks and/or workstations. If there are m tasks and r preference constraints, then there are $m!/2^r$ possible task sequences (Baybars, 1986a). Therefore, it can be time consuming for optimum seeking methods to obtain an optimal solution within this vast search space. Despite the vast search space, many attempts have been made in the literature to solve the ALBP using optimum seeking methods, such as linear programming (Salveson, 1955), integer programming (Bowman, 1960), dynamic programming (Held, Karp, and Shareshian, 1963) and branch-and-bound approaches (Jackson, 1956). However, none of these methods has proven to be of practical use for large problems due to their computational inefficiency. Hence, numerous research efforts have been directed towards the development of heuristics such as Dar-El's MALB (1973), Dar-El and Rubinovitch's MUST (1979), and Baybars' LBHA (1986b) and meta-heuristics such as simulated annealing

S. Ozmehmet Tasan (✉) · S. Tunali
Department of Industrial Engineering,
Dokuz Eylul University, 35100, Bornova, Izmir, Turkey
e-mail: seren.ozmehmet@deu.edu.tr

S. Tunali
e-mail: semra.tunali@deu.edu.tr

(Suresh and Sahu, 1994), tabu search (Peterson, 1993) and genetic algorithms (Falkenauer and Delchambre, 1992).

Among these meta-heuristics, genetic algorithms (GAs) received an increasing attention from the researchers since it provides an alternative to traditional optimization techniques by using directed random searches to locate optimum solutions in complex landscapes. Despite the popularity of GAs' application to ALBP, there exist only a few papers reviewing the subject including Dimopoulos and Zalzala (2000), Aytug et al. (2003) and Scholl and Becker (2006). Dimopoulos and Zalzala (2000) reviewed the use of evolutionary computation methods for solving manufacturing optimization problems including the classic job-shop and flow-shop scheduling problems, assembly line balancing and aggregate production planning. Aytug et al. (2003) reviewed over 110 papers using GAs to solve various types of production and operations management problems including production planning and control, facility layout design, line balancing, supply chain management and etc. We noted that none of these studies placed an adequate amount of emphasis on the use of GAs for solving ALBPs since their scope was very broad. In a recent paper, Scholl and Becker (2006) presented a review and analysis of exact and heuristic solution procedures for solving ALBPs. However, the scope of this study is limited to the review of GAs for particularly solving the simplest version of ALBPs.

In this review, the context is much broader; we focus on solving all types of ALBPs using GAs. A structural framework is provided in order to classify the reviewed papers according to the type of ALBP studied, the GA methodology and the performance specifications. Based on this structural framework, the commonalities and differences between the published studies are discussed and some open issues are identified. Considering the growing number of published literature in this area, we hope that a comprehensive up-to-date review study will guide the researchers about the new promising research directions.

The rest of the paper is organized as follows; In Section "background information" on the main features of ALBPs and GAs are given. Section "Review of published literature", the contributions of reviewed literature are discussed based on various criteria included in the proposed structural framework. Finally, the concluding remarks and future research directions are given in the last section.

Background information

Assembly line balancing

Assembly lines, which consist of a sequence of tasks, each having an operational processing time and a set of precedence relations, are widely adopted in manufacturing plants. Most of the work related to the assembly lines concentrates on ALBP, which deals with the allocation of the tasks among workstations, so that the precedence relations are not violated and a given objective function is optimized. The precedence relations contain the ordering in, which tasks must be performed. The widely used notations in assembly line balancing literature are presented in Table 1.

Figure 1 illustrates the precedence relations by means of a precedence graph, which contains 11 nodes for tasks, node weights in italic for task processing times and arcs for orderings. It is noted that the most commonly used objective function in the literature is the maximization of the line efficiency, $E = t_{sum}/(n^*c)$.

Various classification schemes exist for ALBPs (Baybars, 1986a; Becker and Scholl, 2006; Erel and Sarin,

Table 1 List of notations

Notations	Definitions
n	Number of workstations; i = 1,. . .,n
c	Cycle time
m	Number of tasks; j = 1,. . ., m
t_j	Processing time of task j
t_{sum}	Total processing time of tasks; $t_{sum} = \sum_{j=1}^{m} t_j$
WS_i	Workstation load of workstation i
$t(WS_i)$	Workstation time of workstation i; $t(WS_i) = \sum_{j \in WS_i} t_j$
$maxt(WS_i)$	Maximum workstation time
k	Largest single processing time of a task, a constant
N_v	Number of violations in precedence relations
$\tilde{c}$	Fuzzy cycle time
$t(\tilde{WS}_i)$	Fuzzy workstation times for workstation i
n_s	Number of workstations in solution s
M	Number of models; k = 1,. . ., M
q_k	Demand ratio of model k
it_{ik}	Idle time for workstation i after processing model k
IT_i	Average idle time for workstation i; $IT_i = \sum_{k=1}^{M} q_k it_{ik}$
E	Line efficiency
$f(s)$	Fitness function of a solution s

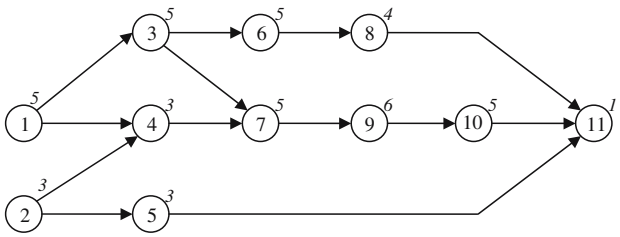

Fig. 1 Precedence graph

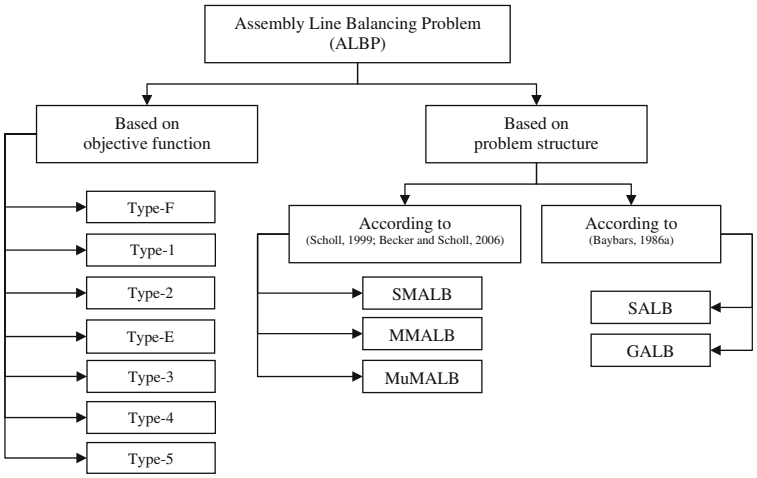

Fig. 2 Classification of assembly line balancing problems

1998; Ghosh and Gagnon, 1989; Scholl, 1999). Figure 2 illustrates the classification of ALBPs based on the objective function and problem structure. This classification compiles Baybars (1986a), Kim et al. (1996), Scholl (1999) and Becker and Scholl (2006).

Several versions of ALBP arise by varying the objective function (Scholl, 1999). Type-F is an objective-independent problem, which is to establish whether or not a feasible line balance exists for a given combination of m and c. Type-1 and Type-2 have a dual relationship; the first one tries to minimize the number of workstations for a given cycle time, and the second one tries to minimize the cycle time for a given number of workstations. Type-E is the most general problem version, which tries to maximize the line efficiency by simultaneously minimizing the cycle time and a number of workstations. Finally, Type-3, 4 and 5 correspond to maximization of workload smoothness, maximization of work relatedness and multiple objectives with Type-3 and Type-4, respectively (Kim et al., 1996).

Based on the problem structure, ALBPs can be classified into two groups. While, the first group (Becker and Scholl, 2006; Scholl, 1999) includes single-model assembly line balancing (SMALB), multi-model assembly line balancing (MuMALB), and mixed-model assembly line balancing (MMALB), the second group (Baybars, 1986a) includes simple assembly line balancing (SALB) and general assembly line balancing (GALB). The SMALB problem involves only one product. The MuMALB problem involves more than one product produced in batches. The MMALB problem refers to assembly lines, which are capable of producing a variety of similar product models simultaneously and continuously (not in batches). Additionally, SALB problem, the simplest version of the ALBP and the special version of SMALB problem, involves production of only one product, where assembly line has features such as paced line with fixed cycle time, deterministic independent processing times, no assignment restrictions, serial layout, one sided workstations, equally equipped workstations and fixed rate launching. The GALB problem includes all of the problems that are not SALB, such as balancing of mixed model, parallel, U-shaped and two-sided lines with stochastic dependent processing times; thereby more realistic ALBPs can be formulated and be solved.

Since balancing of assembly lines is a quite active research area, several comprehensive survey papers on solution methods have been published including Baybars (1986a) that surveys the exact (optimal) methods, Talbot et al. (1986) that compare and evaluate the heuristic methods developed, Ghosh and Gagnon (1989) that present a comprehensive review and analysis of the different methods for design, balancing and scheduling of assembly systems, Erel and Sarin (1998) that present a comprehensive review of the procedures for single-model and multi-mixed-model assembly lines, Rekiek et al. (2002) that focus on optimization methods for the line balancing and resource planning steps of assembly line design, Scholl and Becker (2006) that present a review and analysis of exact and heuristic solution procedures for SALB, and Becker and Scholl (2006) that present a survey on problems and methods for GALB with features such as cost/profit oriented objectives, equipment selection/process alternatives, parallel workstations/tasks, U-shaped line layout, assignment restrictions, stochastic task processing times and mixed model assembly lines.

Genetic algorithms

Optimization problems arise in situations where discrete choices must be made, and solving them amounts to finding an optimal solution among a finite or countable infinite number of alternatives. Many optimization problems are NP-hard. Although the efforts made to solve the optimization problems efficiently have produced important progress in

the last years, there is no universal method. Consequently, there is much interest in approximation algorithms that can find near-optimal solutions within a reasonable computation time. Genetic algorithms, which have been introduced by John Holland (1975) in the 1970s have become increasingly popular among approximation algorithms for finding near optimal solutions to large optimization problems.

GA is a stochastic search method inspired by concepts from Darwinian evolution theory and belongs to a class of meta-heuristic methods known as evolutionary algorithm (EA). As a solution approach, GA has two advantages: (i) GA searches a population rather than a single point and this increases the likelihood that the algorithm will not be trapped in a local optimum since many solutions are considered concurrently, and (ii) GA fitness function may take any form and several fitness functions can be utilized simultaneously. The general idea of a typical GA is best explained by the following scheme:

Step 1: t= 0;	// start with an initial time
Step 2: initpopulation P (t);	// construct an initial population of individuals
Step 3: evaluate P (t);	// evaluate fitness of all individuals of initial population
Step 4: t := t + 1;	// increase the time counter
Step 5: P′ := selectparents P (t);	// select a sub-population for offspring production
Step 6: crossover P′ (t);	// crossover the "genes" of selected parents
Step 7: mutate P′ (t);	// perturb the mated population stochastically
Step 8: evaluate P′ (t);	// evaluate its new fitness
Step 9: P := survive P,P′ (t);	// select the survivors from actual fitness
Step 10: while not do steps 4 through 9	// test for termination criterion (time, fitness, etc.)
Step 11: end GA	// terminate the algorithm

GA is initialized with a population of individuals to start the search process. During the search, candidate solutions (individuals) in the solution space (population) are encoded as symbolic strings, known as chromosomes. A typical GA uses two operators, crossover and mutation, to direct the population towards convergence at the global optimum. The search algorithm analyses and extracts superior evolving information from the search space, and guides the search in a prespecified direction with these operators. Crossover allows solutions to exchange information in a way similar to that used by a natural organism undergoing sexual reproduction. Mutation is used to randomly change the value of single genes within chromosomes. Mutation is typically applied very sparingly. After selection, crossover and mutation are applied to the initial population, a part of the existing population survives to next generation and it forms a new population. This process of GA is continued until a termination criterion is met. A fixed number of generations and some form of convergence are typical termination criteria.

GA is a general method, capable of being applied to an extremely wide range of problems ranging from robotics and finance to marketing and manufacturing. In this article, we particularly placed the emphasis on the use of GAs in solving ALBPs. Therefore, detailed discussion on other GAs and applications are beyond the scope of this paper. The reader can refer to Goldberg (1989), Mitchell (1996) and Coley (2003) for an introduction to GAs. Besides, some recent applications of GAs for solving various manufacturing optimization problems can be found in Cheng et al. (1996, 1999), Dimopoulos and Zalzala (2000), Iyer and Saxena (2004), Martens (2004) and Stockton et al. (2004a,b).

Review of the published literature

In this section, using a structural framework given in Fig. 3, we reviewed the present literature based on specifications of problem, GA and performance. As seen in Fig. 3, problem specifications contain the main features of the problems studied, GA specifications summarize information about the GA methods developed, initialization of the population, chromosome representation, fitness function, genetic operators, selection and survival schemes, feasibility issues, and termination criteria and finally, performance specifications include information about the data sets used to test GA, other solution methods to, which the performance of GA was compared, the computation time and the implementation language. Using this structural framework, in Section "Problem specifications" we focused on the problem specifications of the published literature in chronological order, in Section "Genetic algorithm specifications" we analyzed the GAs in order to draw attention to the commonalities and differences between specifications of GAs, and finally, in Section "Performance specification" we investigated the performance specifications of the researches.

Problem specifications

We referred to the classification given in (Baybars, 1986a) to identify the major trends in types of problems studied. Hence, we reviewed the published literature under two groups: SALB and GALB. Table 2 chronologically lists the reviewed studies based on the type of the problems studied and the objective functions.

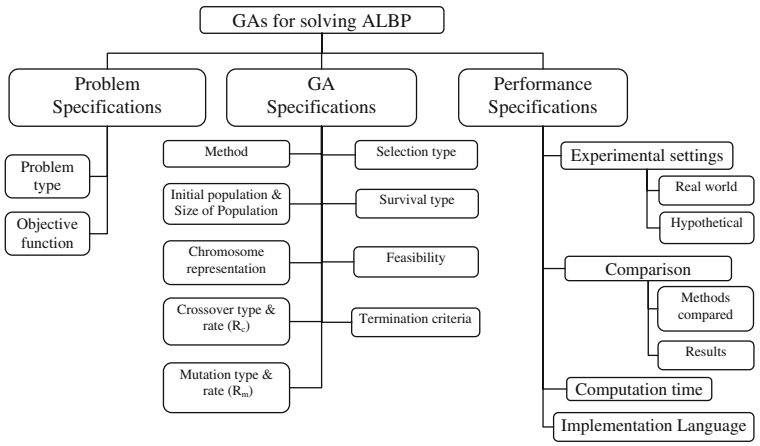

Fig. 3 Structural framework for reviewing

Table 2 Assembly line balancing problem specifications

Problem specifications			
Year	Researcher(s)	Problem type	Objective function
1992	Falkenauer & Delchambre	SALB	Type-1
1994	Leu et al.	SALB	Type-1
1994	Anderson & Ferris	SALB	Type-2
1995	Rubinovitz & Levitin	SALB	Type-2
1995	Tsujimura et al.	GALB (SMALB)	Type-1
1996	Kim et al.	SALB	Type-1, 2, 3, 4, 5
1996	Suresh et al.	GALB (SMALB)	Type-1
1997	Falkenauer	GALB (SMALB)	Type-1
1998	Ajenblit & Wainwright	GALB (SMALB)	Type-1
1998	Chan et al.	GALB (SMALB)	Type-1
1998a	Kim et al.	SALB	Type-2
1999	Rekiek et al.	SALB	Equal Piles
2000	Bautista et al.	SALB	Type-1, Type-2
2000	Kim et al.	GALB (SMALB)	Type-1
2000	Ponnambalam et al.	SALB	Type-1, Type-3
2000	Sabuncuoglu et al.	SALB	Type-1
2001	Carnahan et al.	SALB	Type-2
2001a	Simaria & Vilarinho	GALB (MMALB)	Type-2
2002	Chen et al.	GALB (Assembly planning)	Type-2
2002	Goncalves & De Almedia	SALB	Type-1
2002	Miltenburg	GALB (MMALB & sequencing simultaneously)	Type-1
2002	Valente et al.	GALB (SMALB)	Type-2
2004	Brudaru & Valmar	GALB (SMALB)	Type-1
2004	Martinez & Duff	GALB (SMALB)	Type-1
2004	Simaria & Vilarinho	GALB (MMALB)	Type-2
2004a, 2004b	Stockton et al.	SALB	Type-1
2005	Brown & Sumichrast	SALB	Type-1
2006	Levitin et al.	GALB (SMALB)	Type-2
2006	Noorul Haq et al.	GALB (MMALB)	Type-1

Research on SALB problem

Typical features of SALB problem are one product production, paced line with fixed cycle time, deterministic independent processing times, no assignment restrictions, serial layout, one sided and equally equipped workstations and fixed rate launching.

Falkenauer and Delchambre (1992) were the first to solve SALB problem with GAs. Falkenauer (1991) presented the Grouping Genetic Algorithm (GGA) especially for solving grouping optimization problems, where the aim was to group members of a set into a small number of families in order to optimize objective function under given constraints. GGA has a special chromosome representation scheme and genetic operators, which are used to suit the representation scheme. Later, Falkenauer and Delchambre (1992) implemented the GGA to two grouping optimization problems; i.e. bin packing problem and SALB Type-1 problem. This study was the first attempt to balance an assembly line Type-1 problem with GA. The authors first presented a special representation scheme and special genetic operators for the bin packing problem, and they later modified the special genetic operators for line balancing. Other implementations of GGA for solving ALBPs can be found in Falkenauer (1997), Rekiek et al. (1999) and Brown and Sumichrast (2005). For more information about GGA, the reader may refer to Falkenauer (1998), where he describes the drawbacks, i.e. problems with standard representation scheme and standard genetic operators, which occur when applying the typical GA to grouping problems.

Following Falkenauer and Delchambre (1992), SALB problem was studied by many researchers. Leu et al. (1994) developed a GA to solve SALB Type-1 problems and used heuristic procedures to determine the initial population. They also proposed a number of techniques to deal with the feasibility problems during initialization of the population as well as after the reproduction phase. They also demonstrated the possibility of balancing assembly lines with multiple criteria and zoning constraints.

The first article, which presented a GA application to the SMALB Type-2 problem, was published by Anderson and Ferris (1994). The authors mainly aimed at showing the effective use of GAs solving combinatorial optimization problems. They first described a fairly typical serial implementation of GA for the ALBP and studied the effects of various GA variables on the performance of the GA. Following, they introduced an alternative parallel version of the GA, where each individual in the population resided on a processor. The comparative study between serial GA and parallel GA showed that the quality of the solutions from the parallel implementations was worse than the best solutions obtained from serial implementation.

Rubinovitz and Levitin (1995) used a GA to obtain SALB Type-2 problem in, which the processing times of a task was dependent upon workstation assignment. The authors compared the proposed GA to Dar El and Rubinovitch MUST (1979), where the proposed GA solved the problems involving more than 20 workstations faster than MUST. Finally, the authors concluded that their GA achieved its greater advantage when the precedence constraints were the least restrictive.

Kim et al. (1996) developed a GA to solve multiple objective SMALB problem. They addressed several types of ALBP such as minimize number of workstations (Type-1), minimize cycle time (Type-2), maximize workload smoothness (Type-3), maximize work relatedness (interrelated tasks are allotted to the same workstation as much as possible) (Type-4), and a multiple objective with Type-3 and Type-4 (Type-5). The authors placed the emphasis on seeking a set of diverse Pareto optimal solutions. Although, Kim et al.'s multi-objective GA seems to be very promising, the chromosome representation scheme they used is not well suited to the some of the problem types, since they used a single chromosome representation scheme to represent all of the problem types.

Kim et al. (1998a) considered maximizing the workload smoothness, which has been generally neglected in the literature. Extensive computational experiments were made and the advantages of incorporating problem specific heuristics information into the algorithm were demonstrated. The experimental results showed that the proposed GA outperformed the existing heuristics and the standard GA.

Rekiek et al. (1999) proposed a GGA (Falkenauer and Delchambre, 1992) based on Equal Piles approach for solving SALB problem. They tried to assign tasks to fixed number workstations in such a way that the workload of each workstation was nearly equal by leveling on average the size of each workstation (minimizing the standard deviation of sizes). Therefore, the proposed method warranted to obtain the desired number of workstations and tried to equalize the workloads of workstations as possible. Later, Rekiek developed a GGA for solving multi-objective assembly line design problem in his PhD thesis (Rekiek, 2000).

Bautista et al. (2000) considered the SALB problem with incompatibilities between tasks. To avoid assigning two incompatible tasks to the same station, the authors developed a Greedy Randomized Adaptive Search Procedure (GRASP) obtained from the application of some classic heuristic methods and a GA. They first tried to solve the SALB Type-1 problem and then the SALB Type-2 problem once the number of workstations has been determined. They also revised GRASP by using weights and called it Greedy Randomize Weighted Adaptive Search Procedure (GRWASP). In the proposed method, the greedy heuristic methods were based on

the application of priority rules for assignment of tasks to workstations such as longest processing time and greatest number of immediate successors. The greedy heuristic favors tasks with the best index value, while the GA phase simply changes the order of elements in the solution. Their comparative study showed that the proposed GA and GRWASP resulted in better performance than the greedy heuristics and GRASP.

Ponnambalam et al. (2000) developed a multi-objective GA for SMALB Type-1 problem to optimize several objectives simultaneously: the number of workstations, the line efficiency, and the smoothness index. Several comparisons were made between other heuristics on several examples. The results of the comparisons indicated that GA performed better in all cases studied. However, the execution time for the GA was found to be longer.

Sabuncuoglu et al. (2000) developed a new GA to solve the SMALB problem by utilizing the intrinsic characteristics of the problem. The authors also proposed a method called 'dynamic partitioning' that modified chromosome structure of GAs to save CPU time. The method modifies the chromosome structure by allocating tasks to workstations (i.e. freezing certain tasks) that satisfy some criteria, and continues with the remaining unfrozen tasks. Furthermore, they constructed a new elitism structure adopted from the concept of simulated annealing. It is observed that this new elitism structure contributes significantly to the performance of the GA. In fact, the results of extensive computational experiments indicated that the proposed GA approach outperformed the well-known heuristics in the literature.

Carnahan et al. (2001) considered the physical demands placed on workers in solving SALB Type-2 problem. In order to measure physical demand, the authors used grip strength capacity that represented the maximum finger flexor strength generated by a worker using a semi-pronated power grip. Three methods, i.e. a ranking heuristic, a combinatorial GA and a problem space GA, were developed to simultaneously minimize the maximum manual gripping demands and the cycle time. The authors concluded that the problem space GA performed better than the others.

Goncalves and De Almedia (2002) presented a hybrid GA, which combined heuristic priority rules with GA to solve SALB Type-1 problem. Several problems from the literature have been used to demonstrate the effectiveness and robustness of the proposed hybrid GA. The result of the experiments showed that the proposed method performed remarkably well.

Stockton et al. (2004a,b) investigated the use of GAs for solving various problems that arises when designing and planning manufacturing operations, i.e. assortment planning, aggregate planning, lot sizing within material requirement planning environments, line balancing and facilities layout. In Stockton et al. (2004a), the authors have examined the application of GA to the SMALB Type-1 problem. They compared the performance of the GA with a traditional solution method, i.e. Ranked Positional Weight (RPW) (Helgerson and Birnie, 1961). In Stockton et al. (2004b), the authors performed computational experiments in order to identify suitable genetic operators and parameter values. As these two papers complement each other, they are reviewed together in the following sections.

Brown and Sumichrast (2005) compared the performance of GGA (Falkenauer, 1991) against the performance of typical GA across a range of grouping problems, i.e. bin packing, machine part cell formation and SALB Type-1 problem. They applied the two techniques, i.e. standard GA and GGA, to a set of problems and compared the results with respect to solution quality and computation time. They noted that both of the techniques managed to find the optimal solution for all test problems; however GGA found the optimal solution more quickly.

Research on GALB problem

The studies reviewed in this part focus on GALB problems that include all of the problems that are not SALB, such as balancing of single-model or mixed-model, parallel, U-shaped and two-sided lines with stochastic, fuzzy or dependent processing times.

Tsujimura et al. (1995) were the first to solve GALB problem with GAs. The authors used the fuzzy numbers to represent the imprecise, vague and uncertain task processing times as the processing times are uncertain due to both machine and human factors. They proposed a GA to solve SMALB Type-1 problem, represented the fuzzy processing times by triangular membership functions and illustrated the application of the proposed GA on a problem with 80 tasks.

Following Tsujimura et al. (1995), several versions of GALB problem were studied by many researchers. Suresh et al. (1996) used a GA to solve the SMALB Type-1 problem with stochastic processing times. A modified GA, working with two populations (one allowing infeasible solutions), and exchange of specimens at regular intervals, were proposed for handling irregular search space (i.e. the unfeasibility problem due to problem specifications). The authors believed that a population of feasible solutions would lead to a fragmented search space, thus increasing the probability of getting trapped in a local minimum. They stated that infeasible solutions can be allowed in the population only if the genetic operators can lead to feasible solutions from infeasible ones. Throughout the generations, some solutions were exchanged at regular intervals between the two populations (i.e., 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.

Falkenauer (1997) presented a GA based on GGA (Falkenauer and Delchambre, 1992) and branch and bound algorithm for SMALB Type-1 problem with resource dependent processing times. The problem involved allocating resources with different cost and speed to each task and also assigning the tasks to workstations, in such a way that the total cost of the line be minimal. The author employed GGA to assign the tasks to workstations, and then branch and bound algorithm to select the optimal source for each workstation. In this problem, the processing time of a task depends on the resources being used; therefore, resources with different cost and speed are allocated to each task in addition to the assignment of tasks to workstations, in such a way that the total cost of the line be minimal. In the proposed method, the tasks were assigned to workstations by GGA, and the optimal source for each workstation was selected by branch and bound algorithm.

Ajenblit and Wainwright (1998) were pioneers in balancing the U-shaped SMALB Type-1 problem using GAs. The authors dealt with two possible variations of this problem, minimizing the total idle time and balancing of workload among workstations, or a combination of both. They developed six different assignment algorithms to interpret a chromosome and assign tasks to workstations. The authors applied the proposed GA to 61 test problems. In comparison to previous researchers, they obtained superior results in 11 cases, the same results in 49 cases and worse result in one case.

Chan et al. (1998) proposed a GA for SMALB Type-1 problem in the clothing industry. The authors tried to improve the line efficiency by minimizing the time spent in assembly line balance planning. They also included the various skill levels of workers as problem specific information to solve 41-task ALBP. The experimental results showed that the performance of GA was much better than the performance of the greedy algorithm, which performed optimization by proceeding to a series of alternatives and assigned most skillful worker to each task.

Kim et al. (2000) developed a GA for balancing two-sided SMALB Type-1 problem with positional constraints. Two-sided assembly lines consist of two connected serial lines in parallel, where some task can be performed at one of the two sides of the line, while the others can be performed at the either side of the line. In the two-sided assembly lines, the tasks are classified into three types: L (left); R (right); and E (either) type tasks. L type tasks are easily performed at the left hand side of the line, similarly R type tasks are easily performed at the right hand side of the line and E type tasks are easily performed at both sides of the line. The performance of the proposed GA was compared to integer programming and other heuristic methods at Kim et al. (1998b) using five test problems. The results indicated that the proposed GA showed better performance than the heuristics studied. The authors stated that the proposed GA can be directly applied to the different versions of the ALBP.

Simaria and Vilarinho (2001a) proposed an iterative search procedure including GA for MMALB Type-2 problem with parallel workstations. The proposed GA procedure was originally based on the model developed in Simaria and Vilarinho (2001b) for SMALB Type-2 problem, where the simulated annealing was used as a solution method. The iterative procedure starts with a lower bound of cycle time and successively solves MMALB Type-1 problem by increasing cycle times. Once a feasible solution is found, the procedure employs a GA to decrease the cycle time. Besides minimizing the cycle time, the procedure minimizes the workload balances. The iterative procedure was illustrated using a simple example with two assembly models and 25 tasks.

Chen et al. (2002) presented a GA approach for assembly planning involving various objectives, such as minimizing cycle time, maximizing workload smoothness, minimizing the frequency of tool change, minimizing the number of tools and machines used, and minimizing the complexity of assembly sequences. They classified the assembly line planning problems into line balancing, tooling and scheduling problem. The proposed method was improved by including heuristic solutions into initial population and developing a self tuning method to correct infeasible chromosome. Several examples were employed to illustrate the proposed GA. Experimental results indicated that the proposed GA efficiently yields many alternative assembly plans to support the design and operation of an assembly system.

Miltenburg (2002) solved the assembly line balancing Type-1 problem and sequencing problems simultaneously for mixed model U-shaped assembly line. They proposed a GA to solve the balancing and sequencing problems jointly. The proposed GA was found to offer good solutions.

Valente et al. (2002) proposed a GA to solve assembly line balancing Type-2 problem in a real-world application, a two-sided car assembly line. The solution to the problem involved satisfying the constraint that the length of each workstation was constant. The proposed GA was found to reduce the total assembly time of the current line by 28.5%.

Brudaru and Valmar (2004) proposed a hybrid GA for solving SMALB Type-1 Problem. They considered the processing times of tasks as fuzzy numbers like Tsujimura et al. (1995). Their hybrid method combined the branch and bound with GA. The authors presented a special chromosome representation scheme, embryonic representation, which used subsets of solutions rather than the individual solutions. They also proposed a new type of genetic operator called growing operator to be used for the hybrid GA. The proposed hybrid GA was found to take longer computation time with respect to solution quality.

Martinez and Duff (2004) addressed the U-shaped SMALB Type-1 problem. They first solved this problem using 10 heuristic rules adapted from the simple line balancing problem, such as maximum ranked positional weight, maximum total number of follower tasks or precedence tasks, and maximum processing time, and compared these heuristic solutions with the optimal solutions obtained from previous researches. Following, they modified the Ponnambalam et al.'s GA (2000) and inserted the solutions obtained using these heuristic rules to the initial population. They illustrated the proposed GA using the Jackson's problem (1956). The results showed that the addition of a GA can improve the current solution.

Simaria and Vilarinho (2004) expanded the application of their previous work in Simaria and Vilarinho (2001a), where they proposed an iterative GA based search procedure for MMALB Type-2 problem with parallel workstations. The authors have also conducted a set of computational experiments on a set of generated ALBPs.

Levitin et al. (2006) proposed a GA for solving a special kind of SMALB Type-2 problem, i.e. Robotic Assembly Line Balancing (RALB) problem. The authors defined a robotic assembly line, where robots with different capabilities and specializations were assigned to the assembly tasks. Various procedures for adapting the GA to the RALB problem such as a local optimization (hill climbing) work-piece exchange procedure were introduced. Tests were conducted on a set of randomly generated problems to determine the most effective GA procedure based on the best combination of parameters.

Noorul Haq, Jayaprakash, and Rengarajan (2006) proposed a hybrid GA for solving MMALB Type-1 problem. They incorporated the solution from the modified RPW (MRPW) method into the GA's randomly generated initial population to reduce the search space within the global search space. It was noted that this integration reduced the search time. The authors illustrated the implementation of hybrid GA approach on seven problems and compared the results with the MRPW and the standard GA. The results showed that the proposed approach performed better than the standard GA.

Findings based on problem specifications

Referring to Table 2, we could list the findings of this survey study based on problem specifications as follows:

- Almost half of the articles surveyed (i.e. 14 out of 29) focused on SALB, the simplest version problem of assembly line balancing, while other half focused on GALB.
- Only four of the articles surveyed (Simaria and Vilarinho, 2001a, 2004; Miltenburg, 2002; Noorul Haq et al., 2006) dealt with MMALB problem.
- One of the articles (Miltenburg, 2002) tried to solve balancing and sequencing problems of mixed model assembly line simultaneously.
- 19 out of 29 articles surveyed studied Type-1 problem, minimization of number of workstations. The other nine researches focused on Type-2 problem, minimization of cycle time. Rekiek et al. (1999) considered the equal piles approach. Only Kim et al. (1996), Bautista et al. (2000) and Ponnambalam et al. (2000) considered the multi-objective problem.
- Two of the assembly lines Kim et al. (2000); Valente et al. (2002) studied were two-sided, two of them (Ajenblit and Wainwright, 1998; Martinez and Duff, 2004) were U-shaped, and two of them (Simaria and Vilarinho, 2001a, 2004) had parallel workstations.
- Only one article (Suresh et al., 1996) dealt with stochastic, another two (Brudaru and Valmar, 2004; Tsujimura et al., 1995) dealt with fuzzy, and all the others dealt with deterministic processing times.
- Only one article (Rubinovitz and Levitin, 1995) dealt with workstation dependent, and another one (Falkenauer, 1997) dealt with resource dependent deterministic processing times.
- Only one article (Bautista et al., 2000) considered the incompatibilities between tasks.
- Only one article (Carnahan et al., 2001) considered the physical demands placed on workers during assembly line balancing.
- Only one article (Levitin et al., 2006) considered RALB problem, where robots have different capabilities and specializations.

From the problem specifications perspective, we noted that most of the researchers focused on SALB, the simplest version of the problem, with single objective and ignored the recent trends, i.e. mixed model production, U-shaped lines and etc, in the complex manufacturing environments, where ALBP are multi-objective in nature.

Genetic algorithm specifications

In this section, the proposed GAs were investigated based on nine criteria; the proposed GA method, formation of initial population, the genetic representation of individuals (chromosome), the evaluation function to compute the fitness of individuals (fitness function), genetic operators to alter new individuals (crossover and mutation), selection scheme to select individuals for mating, the survival scheme to generate a new population, feasibility issues and termination criteria. Table 3 presents the chronological order of published literature based on these nine criteria.

Table 3 Genetic algorithm specifications of literature in chronological order

Year	Researcher(s)	Genetic algorithm specifications								
		Method	Initial population & size of population	Chromosome representation	Crossover type & rate (Rc)	Mutation type & rate (Rm)	Selection type (for mating)	Survival type (replacement or reproduction)	Feasibility force/Repair/ Penalty	Termination criteria
1992	Falkenauer and Delchambre	Standard (GGA)	Random	Grouping based & variable length	Modified BPCX	Modified BPM	–	–	Force	Up to 10000 generation
1994	Leu et al.	Standard	Random+ Heuristics & popsize = 20 (growing)	Task based & length = no. of tasks	OX & Rc = 98%	Scramble mutation & Rm = 2%	Roulette wheel	Elitism	Force	Up to 500 generation + converge
1994	Anderson and Ferris	Standard & parallel	Random+ heuristics & popsize = 64	Workstation based & length = no. of tasks	One point crossover & Rc = 60–70–80%	One point mutation & Rm = 0.5–4%	Stochastic universal sampling	Elitism	Penalty	Up to 350 generation + converge
1995	Rubinovitz and Levitin	Hybrid GA	Random	Task based & length = no. of tasks	Fragment reordering crossover (FRG)	FRG mutation (FRGm)	Randomly	Elitism	Force	Up to T generation
1995	Tsujimura et al.	Standard	Random	Task based & length = no. of tasks	PMX	Swap mutation	Elitism	Elitism	Repair	–
1996	Kim et al.	Standard	Random & popsize = 100	Task based & length = no. of tasks	Standard and non standard crossover & Rc = 40-60%	Standard and non standard mutation & Rm = 20-40%	Tournament	Elitism	Repair	-
1996	Suresh et al.	Standard	Random & popsize = 40-60 (with 2 population)	Workstation based & length = no. of tasks	One point crossover & Rc = 50-70%	Interchange mutation & Rm = 1%	Elitism	Elitism	Repair in std GA & penalty in 2 pop GA	–
1997	Falkenauer	GGA & Branch and Bound	Random	Grouping based & variable length	Modified BPCX	Modified BPM	–	–	Force	–
1998	Ajenblit and Wainwright	Standard	Random & popsize = 100	Task based & length = no. of tasks	OX	Not used	–	–	Force	–
1998	Chan et al.	Standard	Random & popsize = 50	Task based & length = no. of tasks	Uniform (uniform order- based)	SSM	Roulette wheel	Survive children	Force	Terminate at 5000 s
1998a	Kim et al.	Heuristic based GA	Random & popsize = 100	Workstation based & length = no. of tasks	HSX & Rc = 70–90%	HSM & Rm = 10–20 %	Tournament	Elitism	Force	Converge
1999	Rekiek et al.	Hybrid GGA	Random	Grouping based & variable length	Modified BPCX	–	–	–	Force	–
2000	Bautista et al.	Heuristic based GA	–	Heuristic based & length = no. of heuristics	–	–	–	–	No need	Up to T generation
2000	Kim et al.	Standard	Random+ Heuristics	group number	Structured one point crossover (SOX) & Rc = 50%	Random mutation & Rm = 10%	Tournament	Elitism	–	Up to T generation
2000	Ponnambalam et al.	Standard	Random	Heuristic based & length = 14 (no. of heuristic)	Two point crossover	Random mutation	Roulette wheel	Elitism	No need	-
2000	Sabuncuoglu et al.	Standard	Random	Task based & length = no. of tasks	Order crossover	Scramble mutation	Roulette wheel	Elitism	Force	Up to T generation
2001	Carnahan et al.	Hybrid GA	Random & popsize = 60	Task based & length = no. of tasks	FRG & Rc = 60%	FRGm	Roulette wheel	Elitism	Force	Up to T generation + converge

Table 3 continued

Year	Researcher(s)	Genetic algorithm specifications								
		Method	Initial population & size of population	Chromosome representation	Crossover type & rate (Rc)	Mutation type & rate (Rm)	Selection type (for mating)	Survival type (replacement or reproduction)	Feasibility force/Repair/ Penalty	Termination criteria
2001a	Simaria and Vilarinho	Two staged iterative GA	Random+ Heuristics	Workstation based & length = no. of tasks	SOX	One point mutation	Tournament	Elitism	Repair	Up to T generation + converge
2002	Chen et al.	Standard	Random + Heuristics	Workstation based &	Order1-Order2-PMX-Cycle	Swap mutation	Roulette wheel	Elitism	Repair using self-tuning	–
2002	Goncalves and De Almedia	Standard & hybrid with heuristic priority rules	Random+ Heuristics & popsize = no. of tasks	Random key heuristic based & length = no. of tasks	Uniform crossover & Rc = 70 %	Randomly generate & Rm = 20%	Copy 15%	Elitism	No need	Up to (3*no. of tasks)
2002	Miltenburg	Standard	Random & popsize = 50	Combination of task based and model sequence based & length = no. of tasks+ model numbers	OX and Cycle	Swap mutation	Rank selection with elitism	Elitism	Repair	Terminate at 300 s
2002	Valente et al.	Standard	Random & popsize = 100	Workstation based & length = 13(no. of tasks)	One point crossover & Rc = 80%	Simple bit mutation & Rm = 4 %	Stochastic universal sampling	Elitism	Penalty	Up to 200 generation
2004	Brudaru and Valmar	GA & Branch and Bound		Embryonic & variable length	–	–	–	–	–	–
2004	Martinez and Duff	Standard	Random & popsize = 20	Heuristic based & length = 10 (no. of heuristic)	–	–	–	–	No need	–
2004	Simaria and Vilarinho	Two staged iterative GA	Random + Heuristics	Workstation based & length = no. of tasks	SOX	One point mutation	Tournament	Elitism	Repair	Up to T generation + converge
2004a, 2004b	Stockton et al.	Standard		Binary	Two point crossover & Rc = 60-65-70-75 %	Rm = 5–2.5–0.5%	Roulette wheel	Elitism	Penalty	–
2005	Brown and Sumichrast	Standard (GGA)	–	Grouping based & variable length	Modified BPCX	–	–	–	Force	–
2006	Levitin et al.	Hybrid GA	Random & popsize = 100	Task based & length = no. of tasks	FRG	Swap mutation & Rm = 1 %	Randomly	Elitism	Force	Up to T generation
2006	Noorul Haq et al.	Hybrid GA	Random + Heuristic	–	Rc = 80 %	Rm = 5 %	–	–	–	–

Considering the great number of criteria investigated, we summarized the findings of this review study under two parts; in the first part, we considered only six of the criteria including the GA method, initial population, selection scheme, survival scheme, feasibility issues and termination criteria. The second part included the discussion of published literature based on chromosome representation scheme, genetic operators and fitness evaluation.

Many of the GAs developed to solve ALBPs have common characteristics. As it is seen in Table 3, most of the researchers (19 out of 29) used a typical GA procedure (see Section "Background information"). It must be noted that Falkenauer and Delchambre's GGA (1992), which was implemented in four researches, was also considered as a typical GA, since it consisted of a new representation scheme and special genetic operators without changing the general procedure of a typical GA. Among the papers surveyed, only Anderson and Ferris (1994) employed both the typical GA and also parallel GA. We noted the combination of GA with branch and bound in two studies: Falkenauer (1997) employed the GGA for assignments of tasks and branch and bound for selection of optimal resource, and Brudaru and Valmar (2004) combined the GA with branch and bound to evaluate the fitness function. A recent trend to increase efficiency of GAs is to hybridize them with heuristics, Kim et al. (1998a), Bautista et al. (2000) and Goncalves and De Almedia (2002). Lastly, to solve Type-2 problems, Simaria and Vilarinho (2001a, 2004) proposed a two staged iterative GA by sequentially evaluating the solutions of Type-1 and then Type-2 problem.

A GA starts from a set of individuals called initial population. Numerous initialization methods exist ranging from random methods that produce an entirely random initial population to more direct methods that produce a selective initial population when prior knowledge of the search space is known. There are also other initialization methods, which combine both of these approaches; for example, initialization by inserting the results of heuristic solutions to the problem. Inclusion of heuristic generated solutions to the population was first reported by Leu et al. (1994) and Anderson and Ferris (1994). It is noted that most of the researchers (i.e., 16 out of 29) generated the initial populations randomly. The other researchers used both randomly and heuristic generated individuals in the initial population. To initialize the population, Ponnambalam et al. (2000) used the 14 simple greedy heuristics given at Talbot et al. (1986).

After generation of the initial population, first, the fitness of each individual in the population is calculated, the potential parents are selected to create the offspring (new individuals) and then a selection scheme is used to select individuals for mating based on their relative fitness. With some exceptions (see Table 3, most of the GAs in literature used either roulette wheel selection or stochastic universal sampling to select the parents. In roulette wheel selection, a sector of a roulette wheel whose size is proportional to the appropriate fitness measure is assigned to the individuals, then a random number is generated (spin the wheel), and the parents are selected according their random position on the wheel. In stochastic universal sampling, an individual is selected entirely on its position in the population and its relative distance to the other individuals in the population. Hence, this scheme prevents unfit individuals from dominating the selection process.

In a GA, survival is an essential process that removes individuals with a low fitness and drives the population towards better solutions. Survival scheme is tightly related to the size of the population. Most of the current GAs for ALBP assumes a constant population size N, which is a user-controlled input parameter. The GAs with constant population size is generally called as the "steady-state" GA. This kind of GA rigidly enforces this limit (N), in the sense that each time an offspring is produced resulting N + 1 individuals, a survival scheme is invoked to reduce the population size back to N. By contrast, Leu et al. (1994) permitted more elasticity in the population by allowing the population to grow before a survival scheme is invoked. With some exceptions, all of the researchers used elitism strategy for survival of individuals to next generation. Elitism strategy guarantees that the best individuals of the population survive into the next generation. Chan et al. (1998) modified the elitism strategy; in a way that the parent was replaced with offspring rather than the worst individual in the population was replaced with offspring.

Another important issue in designing the GA is to decide on whether infeasible individuals should be allowed in the population or not. In an ALBP, assigning each task to exactly one workstation and satisfying the precedence relations and all other constraints generates a feasible individual in the GA. However, crossover and mutation operations may result in formation of infeasible individuals. In this study, we noted that the researchers have employed three types of strategies to cope with infeasible individuals such as forcing individuals for feasibility, repairing infeasible individuals and including penalty for infeasible individuals. The first strategy, forcing individuals for feasibility, requires generating only valid individuals in the initial population (Ajenblit and Wainwright, 1998; Chan et al., 1998; Leu et al., 1994). Each gene of the chromosome in the initial population is obtained randomly choosing the next task among unselected tasks whose predecessors had already been chosen. Also, some special genetic operators (see Section "Genetic operators"), which ensure feasible chromosomes, can be used to force individuals for feasibility. An appropriate representation scheme in conjunction with carefully designed genetic operators and fitness function is essential for maintaining the feasibility of chromosomes. The second strategy repairs the infeasible individuals by rearranging the tasks according to precedence relations and other defined constraints (Kim et al.,

1996; Tsujimura et al., 1995). The third strategy leaves out the infeasible individual in the population and calculates its fitness value using a penalty function. Anderson and Ferris (1994) and Stockton et al. (2004a,b) used traditional standard genetic operators, and stated that maintaining a percentage of infeasible individuals in the population would lead to covering a larger area in the search space. Moreover, they added a penalty cost to the fitness function and stated that allowing infeasible individuals to stay in the population would increase the amount of variability in the population. An example of a fitness function with a penalty cost is given in Section "Fitness evaluation".

A GA for ALBP is terminated either after reaching a specified number of generations or after reaching to an acceptable convergence that is usually represented by no improvement in the best solution after a certain amount of generation. The best individuals of the population are then tested to determine if they satisfy the precedence constraints and other zoning restrictions. In this survey study, we noted that most of the researchers specified a maximum number of generations as a terminating condition.

Furthermore, in order to implement GA, it is necessary to construct a chromosome representation scheme to represent the chromosomes, genetic operators to crossover and mutate, and an external objective function to evaluate the fitness. The findings of this review study regarding these criteria are given in the following section.

Chromosome representation scheme

The first step in applying GA to a particular problem is to convert the solutions (individuals) of ALBP into a string type structure called chromosome. This representation must uniquely map the chromosome values (genotypes) onto the decision variable domain.

Alternative chromosome representation schemes will be illustrated using the example given in Fig. 4, where the cycle time c, is 10 min and number of workstations, n is 5. The workstation loads for this solution are $WS_A = \{1, 3\}$, $WS_B = \{2, 4, 5\}$, $WS_C = \{6, 7\}$, $WS_D = \{8, 9\}$, and $WS_E = \{10, 11\}$.

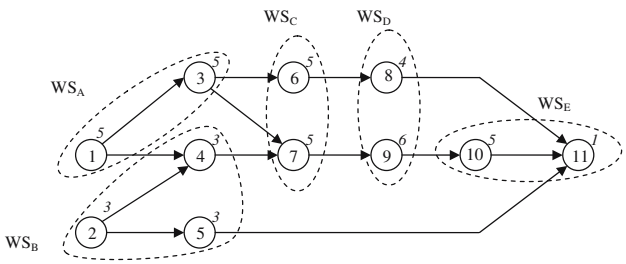

Fig. 4 Solution for c = 10 and m = 5

Real-valued representations are much more effective for the representation of balancing problem. In Scholl and Becker (2006), where only the literature related to SALB was reviewed, four types of chromosome representation schemes are presented. In this survey study covering a broad range of literature on assembly line balancing, we noted six different types of chromosome representation schemes; i.e. task based, embryonic, workstation based, grouping based, binary and heuristic based; each having pros and cons concerning the type of applicable genetic operators. Note that the classic binary representation of the simple GA has only been used for the ALBP in Stockton et al. (2004a,b). Due to the lack of information on binary representation of an individual, in this section, only the remaining five chromosome representation schemes are explained in detail. The chromosome representation schemes are named in order to suit the characteristics of ALBP. These representation schemes can be classified as follows:

1. *Task based representation*: The chromosomes are defined as feasible precedence sequences of tasks (Ajenblit and Wainwright, 1998; Leu et al., 1994; Sabuncuoglu et al., 2000). The length of the chromosome is defined by the number of tasks. For example, the task based representation of the solution given in Fig. 4 is illustrated in Fig. 5a. In order to calculate the fitness of a task based chromosome, additional operations, which assign the tasks to workstations according to the task sequence in the chromosome, is needed. Task based representation is the most appropriate representation for ALBP Type-1, since Type-1 problems consider the minimization of workstations as an objective function.
2. *Embryonic representation*: Embryonic chromosome representation that was proposed by Brudaru and Valmar (2004) is actually a special version of the task based chromosome. Only difference between the two is that the

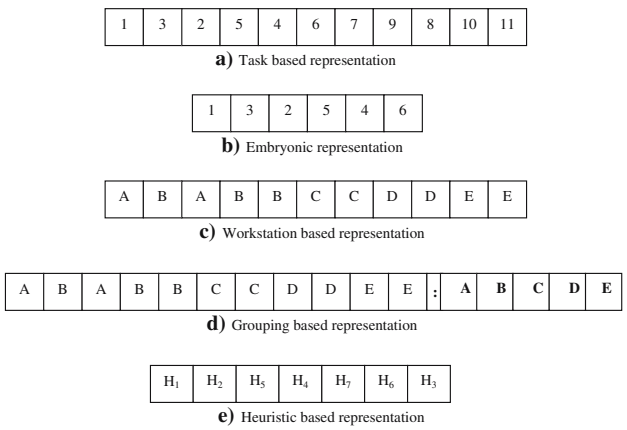

Fig. 5 Chromosome representation schemes used for ALBP

embryonic representation of a solution considers the subsets of solutions rather than the individual solutions. During the generations, the embryonic chromosome evolves through a full length solution. Therefore, the chromosome length varies throughout the generations. The length is initially defined by a random number and then increases until it reaches the number of tasks. Figure 5b illustrates an example of embryonic representation of the solution given in Fig. 4.

3. *Workstation based representation*: The chromosome is defined as a vector containing the labels of the workstations to, which the tasks are assigned (Anderson and Ferris, 1994; Kim et al., 2000). The chromosome length is defined by the number of tasks. For example, the workstation based representation of the solution given in Fig. 4 is illustrated in Fig. 5c, where the task 4 is assigned to workstation B. This kind of chromosome representation scheme is generally used for ALBP Type-2.
4. *Grouping based representation*: This type of representation was proposed by Falkenauer and Delchambre (1992) especially for grouping problems, i.e. ALBP Type-1. The authors stated that the workstation based representation, which is object oriented, is not suitable for ALBP Type-1. In grouping based representation, the workstations are represented by augmenting the workstation based chromosome with a group part. The group part of the chromosome is written after a semicolon to list all of the workstations in the current solution (see Fig. 5d). The length of the chromosome varies from solution to solution. As it is seen in Fig. 5d, the first part is the same as in workstation based chromosome. Difference comes from the grouping part, which list all the workstations, i.e. A, B, C, D, and E.
5. *Heuristic based* (*indirect*) *representation*: This type of representation scheme represents the solutions in an indirect manner. In Goncalves and De Almedia (2002), and Bautista et al. (2000), the authors first coded the priority values of the tasks (or a sequence of priority rules), then they applied these rules to the problem to generate the solutions. The chromosome length is defined by the number of heuristics. For example, Fig. 5e shows an example chromosome having seven different heuristics, which are used in the sequence of H_1, H_2, H_5, H_4, H_7, H_6 and H_3 to assign the tasks to the workstations.

An appropriate chromosome representation scheme in conjunction with carefully designed genetic operators and fitness function is essential for GA design, since the application of standard crossovers or mutations to task, workstation and grouping based chromosomes may result in infeasible solutions. This aspect must be dealt with by penalizing infeasibilities, rearranging the solution by certain heuristic strategies or constructing of special genetic operators, which will force feasibility. In this context, heuristic based chromosomes have the advantage to achieve the feasibility without such difficulties.

Fitness evaluation

In order to mimic the natural process of the survival of the fittest, the fitness evaluation function assigns a value reflecting the relative superiority (or inferiority) to each individual. The objective function provides a measure of an individual's performance or fitness in the search space.

As mentioned in Section "Background informmation", different objective functions, with a range of complexities, are developed for each specific problem type in assembly line balancing. Among these objective functions, we noted an implementation difficulty with the objective of Type-1 as the fitness function. When there are alternate optimal solutions having the same objective value, this objective function does not give a strong distinction between the alternate solutions. This problem is less relevant for Type-2 or Type-E.

During this literature survey, some of following fitness functions have been noted:

- Falkenauer and Delchambre (1992), and Brown and Sumichrast (2005) used the following fitness function, $f(S)$, for SALB Type-1 problem.

$$f(S) = \sum_{i=1}^{n} (c - t(WS_i))^2/n$$

- Anderson and Ferris (1994) used the following fitness function, $f(S)$, which sums up the maximal workstation time and a penalty term for solving SALB Type-2 problem. This penalty term is used to assign some penalty cost to any individual, which is infeasible because of precedence violations.

$$f(S) = \max t(WS_i) + kN_v$$

- Leu et al. (1994) used the following three fitness functions, $f_1(S)$, $f_2(S)$, and $f_3(S)$, for SALB Type-1 problem. The first objective function aims the minimization of mean squared idle time, the second one aims the minimization of mean idle time and the third one combines the first two objectives.

$$f_1(S) = \sum_{i=1}^{n} (c - t(WS_i))^2/n$$
$$f_2(S) = \sum_{i=1}^{n} (c - t(WS_i))/n$$
$$f_3(S) = 2\sqrt{f_1(S)} + f_2(S)$$

- Tsujimura et al. (1995) used balance delay as fitness function, $f(S)$, for solving SMALB Type-1 problem where the cycle time and processing times of tasks are represented by fuzzy numbers.

$$f(S) = \sum_{i=1}^{n} (\tilde{c} - t(\underset{i}{\widetilde{W}}S))$$

- Bautista et al. (2000) used the following fitness function, $f(S)$, which considers the number of workstations in this solution in the first term, the lower bound of number workstations in the second term and the degree of imbalance in the third term.

$$f(S) = -n_s + \left\lceil \frac{\sum_{i=1}^{n} t_i}{c} \right\rceil^{+} + \sqrt{\frac{\sum_{i=1}^{n} (c - t(WS_i))^2}{c \times \sqrt{n_s}}}$$

- Sabuncuoglu et al. (2000) used the following fitness function, $f(S)$, which aims at reducing the imbalance in the first term and minimizing the number of workstations in the second term, for solving SALB Type-1.

$$f(S) = \sqrt{\frac{\sum_{i=1}^{n} (\max(t(WS_i)) - t(WS_i))^2}{n}} + \frac{\sum_{i=1}^{n} (\max(t(WS_i)) - t(WS_i))^2}{n}$$

- Simaria and Vilarinho (2001a, 2004) used the following fitness function, $f(S)$, which considers the cycle time in the first term and balance workstation load in the second term, for solving MMALB Type-2 problem.

$$f(S) = c + \frac{M}{n(M-1)} \sum_{i=1}^{n} \sum_{k=1}^{M} \left(\frac{q_k it_{ik}}{IT_i} - \frac{1}{M} \right)^2$$

Genetic operators

Genetic operators are of two kinds, crossover and mutation. Crossover is the operation by, which two individuals in the current population create offspring for the next population. The mutation is used to maintain diversity in the population. Mutation does this by randomly changing elements in a chromosome.

The GA applications have created a great variety of chromosome representation schemes and genetic operators. The most frequently used are representations for numeric domains, permutation domains, matrix domains, and function domains. It is beyond the scope of this survey study to describe all of them in detail. Here, we will focus on the numeric and function domains, which are the two primary domains relevant for assembly line balancing.

Anderson and Ferris (1994) used traditional standard genetic operators and believed that maintaining a percentage of infeasible individuals in the population can help to obtain a good coverage of the search space. Hence, they included a penalty cost to the evaluation function (see Section "Fitness evaluation").

In this survey, we noted several specialized crossover operators (crossover and mutation), which ensure the feasibility of chromosomes. Some of the specialized crossover operators employed include *Modified Bin Packing Crossover (Modified BPCX)* (Falkenauer and Delchambre, 1992), *Order-Based Crossover (OX)* (Leu et al., 1994), *Modified Partially Mapped Crossover (Modified PMX)* (Tsujimura et al., 1995), *Heuristic Structural Crossover (HSX)* (Kim et al., 1998a), *Uniform Order-Based Crossover* (Chan et al., 1998), and *Order Crossover* (Sabuncuoglu et al., 2000).

Falkenauer and Delchambre (1992) used *modified BPCX* especially generated for grouping based representations. This crossover operator is applied only to the group part of the chromosome, while the workstation based part of the chromosome remains the same. Since the length of the group part of grouping based chromosome varies, this operator can handle variability of chromosome length. Leu et al. (1994) used a modified version of standard two-point order crossover, i.e. *OX*, which cuts each of the parent chromosomes into three parts. One of the offsprings keeps the first and the last part of the first parent. The middle part of the sequence is filled in by adding the missing tasks in the order in, which they are contained in the second parent. The other offspring is built analogously based on the first and the last part of the second parent. Both of the resulting offsprings are feasible due to filling in the middle part in a precedence feasible order. Also, Tsujimura et al. (1995) modified the standard two point order crossover (*modified PMX*). They first exchange the middle parts of the chromosome between parents to create offspring. Then they repaired the resulting chromosome without changing the middle part according to the precedence relations. *HSX* uses problem-specific constraints to choose multiple groups of workstations to copy from two parents to a child. This procedure is repeated twice; therefore, two parents are used to create two children (Kim et al., 1998a). *Uniform order-based crossover* uses two parents, P1 and P2, to create two offspring C1 and C2. A gene string that is the same length as the parent is generated. If the value in gene i of the chromosome is equal to 1, then the value in gene

i in P1 is copied into C1. A list of the elements in P1 that correspond to zeros in the gene string are then permuted so that they appear in the same order that they appear in P2, then the gaps in C1 are filled in with these ordered permuted elements. C2 is created in a similar fashion (Chan et al., 1998). *Order crossover* (Sabuncuoglu et al., 2000) is similar to OX crossover. Two random cut locations are generated for two parents; however, with the Order crossover, the centre section is permuted while the extreme ends remain unchanged. This centre section of each of the two children is generated by filling in the missing elements in the order that they appear in the other parent without duplicating any of the elements within the children's chromosomes.

Specialized mutation operators include *Modified Bin Packing Mutation* (*modified BPM*) (Falkenauer and Delchambre, 1992), *Heuristic Structural Mutation* (*HSM*) (Kim et al., 1998a), *Scramble Sublist Mutation* (SSM) (Chan et al., 1998), and *Scramble Mutation* (SM) (Sabuncuoglu et al., 2000).

Falkenauer and Delchambre (1992) used *modified BPM*, especially generated for grouping based representations. Like crossover operator, this mutation operator is also applied only to the group part of the chromosome, while the workstation based part of the chromosome remains the same. *HSM* (Kim et al., 1998a) randomly selects some tasks from each chromosome, proportional to the mutation rate, and then reassigns those tasks. *SSM* selects a sublist from a parent, by having the centre area created by randomly determining two cut points. Elements within the sublist are then permuted, or scrambled; therefore, the newly created child has the permuted centre section surrounded by the original data made up of the two extreme endpoints of the parents. Similarly, *SM* identifies a cut point in a parent, and the child is created by copying the elements before the cut point into the child, in the order that they appear in the parent. The remaining genes located after the cut are then placed in the child randomly while maintaining feasibility. Surprisingly, Ajenblit and Wainwright (1998) did not use mutation in their implementation. They stated that the mutation operator does not have any potential to improve the solutions since it just replicates the work done by the initialization.

In addition to crossover and mutation operators, Brudaru and Valmar (2004) proposed a new type of genetic operator called growing operator for their embryonic representation scheme. The proposed growing operator helps the subset of a solution represented by embryonic chromosome to evolve through a full length chromosome.

We noted that, when infeasible individuals were allowed in the population, the traditional standard genetic operators are used and on the other hand, allowing only feasible individuals in the population necessitated using special genetic operators. As a consequence, for effective GA design, the researcher must consider genetic operators together with chromosome representation scheme and fitness function.

Performance specifications

In this section, the published literature has been reviewed based on the experimental settings in, which the proposed GAs implemented, the other solution methods to, which the GA's performance compared, the computation time required and, finally, implementation language employed. However, the reader should bear in mind that, as the reviewed GAs employed different ALBP types, test sets and hardware, the comparison among their performances can not be justified. Therefore, this section does not try to compare the performances of the reviewed GAs; it only includes the findings of the comparative studies reported in each paper. For the performance comparison among reviewed GAs, further comparisons must be made under fair conditions and with challenging test sets.

As seen in Table 4, most of the researches evaluated the performance of the proposed GAs using hypothetical problems, such as randomly generated problems (Anderson and Ferris, 1994) and non benchmark problems (Kim et al., 1996; Leu et al., 1994; Suresh et al., 1996). The real-world applications of GAs are only seen in Chan et al. (1998) in clothing industry and Valente et al. (2002) in automotive industry. Furthermore, it is noted that, the computational testing of most GAs has been done either by ignoring existing benchmarks or by using the simple test data. Hence, the findings of the GA researches are limited to a narrow scope. To attract the attention of practitioners to the use of GAs, we believe that it is necessary to divert more effort to solving real-world ALBPs.

For the computational testing, the researchers used several problems with various sizes. For instance, Falkenauer and Delchambre (1992) used a problem with the size of 64 tasks, Leu et al. (1994) used problems with the size of 45, 50 and 100 tasks, and Miltenburg (2002) used problems with the size of 45 and 83 tasks. Generally, we noted that the computational time increases with the size of the problem, indicating the fact that the performance of the GAs is affected by the complexity of the problem.

As for the comparative studies for performance evaluation (see Table 4), the GAs developed for ALBP either outperformed or matched comparative heuristics, such as rank positional weight heuristic, Kilbridge and Wester's heuristic and Hoffmann precedence matrix procedure. Note that, in these comparative studies, only a few number of heuristics was used as benchmark and also little attention has been given to recently introduced heuristics. For comparison, only, Ajenblit and Wainwright (1998) used both dynamic programming and various heuristic algorithms, which were proposed in Miltenburg and Wijngaard's research (1994). In this study, the proposed GA is found to give the same results in 42 problems, superior in 11 problems and worse in 1 problem.

Table 4 Performance specifications of literature in chronological order

Year	Researcher(s)	Performance specifications					
		Experimental settings		Comparison		Computation time	Implementation language
		Real world	Hypothetical	Methods compared	Results		
1992	Falkenauer and Delchambre	–	One problem	Heuristic	Better	300 s	–
1994	Leu et al.	–	Benchmark	Heuristics	Better	13.33 s	–
1994	Anderson and Ferris	–	Randomly generated	–	–	–	–
1995	Rubinovitz and Levitin	–	Randomly generated	MUST	Better	–	–
1995	Tsujimura et al.	–	One problem	–	–	–	–
1996	Kim et al.	–	Benchmark	Heuristics	Promising	Shorter	C++
1996	Suresh et al.	–	Benchmark	Versions of GA	Better	–	–
1997	Falkenauer	–	Randomly generated	–	–	600 s 120 s 1200 s	–
1998	Ajenblit and Wainwright	–	Benchmark	Dynamic prog. & Heuristics	Same in 49, superior in 11 and worse in 1	–	LibGA
1998	Chan et al.	Clothing industry	–	Greedy Algorithm heuristic	Outperformed	GA trials run for 5000 s	–
1998a	Kim et al.	–	Benchmark	Three heuristics & another GA	Mostly outperformed	–	C++
1999	Rekiek et al.	–	One problem	–	–	120 s at most	–
2000	Bautista et al.	–	Randomly generated	Heuristics, GRASP& GRWASP	Better	–	–
2000	Kim et al.	–	Benchmark	Integer prog & heuristics	–	Shorter	–
2000	Ponnambalam et al.	–	Benchmark	Heuristics	Better	Longer	C++
2000	Sabuncuoglu et al.	–	Benchmark	Six heuristics & some known optimal solutions	Performs better than four and matches two	Shorter	–
2001	Carnahan et al.	–	Benchmark	Two heuristics	Better	–	–
2001a	Simaria and Vilarinho	–	One problem	–	–	–	–
2002	Chen et al.	–	Various problems	Heuristics	Better	–	LibGA
2002	Goncalves and De Almedia	–	Benchmark	Heuristics	Outperformed	–	Visual basic 6
2002	Miltenburg	–	Benchmark	Versions of GA	Good	130 s 300 s 300 s	Visual Basic 6
2002	Valente et al.	Automotive industry	–	–	Reduction of time by 28.5%	0.85 s	ANSI C (based on GALOPPS)
2004	Brudaru and Valmar						
2004	Martinez and Duff	–	One problem	Heuristics	–	–	–
2004	Simaria and Vilarinho	–	Randomly generated	–	–	Acceptable	Visual C++
2004a, 2004b	Stockton et al.	–	One problem	RPW	same	–	–
2005	Brown and Sumichrast	–	Benchmark	Standard GA	same	Shorter	–
2006	Levitin et al.	–	Randomly generated	Branch and Bound	better	Shorter	–
2006	Noorul Haq et al.	–	Randomly generated	Modified RPW	better	Shorter	C++

We noted that the computation time (i.e. CPU time), which indicates the efficiency of the proposed GA, was reported by a few researchers; most of them preferred to state it verbally. Based on the reported CPU times, we can state that except for Ponnambalam et al. (2000), who reported an increase in CPU time due to the increased number of generations, most of the researchers obtained a solution in shorter CPU time. In Falkenauer (1997), the optimal solution was found in 600 s for a problem with 30 tasks, five resources/task and 10 workstations, 120 s for a problem with 30 tasks, four resources/task and 10 workstations, and 1200 s for a problem with 40 tasks, four resources/task and 10 workstations, but it must be noted that no precedence relations existed for these randomly generated problems. Among the researchers surveyed, Miltenburg (2002) gave detailed information regarding the performance of the proposed GA; in this study, the average computation times per instance were found to be 130 s when the proposed GA employed two point crossover, 300 s when the proposed GA involved cycle crossover and 300 s when the proposed GA included randomly generated solutions.

Lastly, as seen in Table 4, most of the researchers preferred C++ as an implementation language. This can be attributed to the advantages of C++ in increasing the speed of execution of the programs, and using the memory more effectively by employing various object oriented programming (OOP) concepts, constructors, call by reference, pointers, dynamic memory allocation (new and delete) functions. Falkenaure's GGA (Falkenauer, 1998) was used to develop a commercial software package, i.e. Optiline, which optimizes the line, yielding a detailed assignment of tasks to workstations and operators. It must be noted that except for Falkenaure's GGA (Falkenauer, 1998), all other GAs reported in this survey study were employed for research purposes.

Results and discussion

Genetic algorithms are increasingly being used to solve manufacturing optimization problems. A well-known manufacturing optimization problem is the assembly line balancing problem, which deals with the allocation of the tasks among workstations so that a given objective function is optimized. As GAs have established themselves as a useful optimization technique in the manufacturing field, the application of GAs to assembly line balancing has expanded a lot. Considering the growing number of publications in this area, we thought that this review article will help both the researchers and practitioners to comprehend the current research issues and also provide them with a guideline about future research directions. To identify the current research issues, in particular, we summarized the main specifications of the problems studied, and discussed the proposed GAs with respect to chromosome representations, genetic operators and the fitness functions used for performance evaluation.

From the assembly line balancing perspective, we noted that most of the researchers focused on a simple version of problem with single objective and ignored the multi-objective nature of the problem. As for the GA perspective, it is noted that two important issues have been extensively studied. One is how to encode a solution of the problem into a chromosome and the other is reproduction of new individuals by using genetic operators.

Because of the existence of complex constraints inherent in the problem, a simple binary string does not work at all, since it certainly yields to infeasible or even illegal solutions. The early efforts of most researchers have been devoted to the invention of a new and efficient representation scheme, (i.e. task based, workstation based, grouping based and heuristic based), for the problem. Somewhat surprising, the population initialization has not received much attention so far.

Some researchers proposed new genetic operators to ensure the feasibility of individuals for a certain representation scheme. Another group of researchers claimed that infeasible solutions have to occupy a certain space in the population; hence they used standard genetic operators, which may result in infeasible individuals of a certain representation scheme. An appropriate chromosome representation scheme in conjunction with carefully designed genetic operators and fitness function is essential for GA design.

Besides the ease in adapting to the ALBP, the efficiency of the proposed GAs depends greatly on various control parameters such as the population size, the probability of crossover and mutation. It is noted that most of the researchers did not give enough attention to optimization of parameters to control evolutionary process effectively. Since the information about the control parameters is not published in detail, it is difficult to compare the performance of the various techniques. Moreover, because of the lack of clear and consistent reporting of parameters, procedures, and results, it is unlikely that other researchers could reproduce these works.

In most of the published literature, GAs are found to be competitive to the best known constructive methods. However, the findings of these researches are limited to a narrow scope, since the computational testing of the reported GAs has been done by ignoring existing test beds and state-of-the-art solution methods or by using the most simple test data available. Moreover, based on the published researches, it is impossible to make a global comparative evaluation of the different GAs proposed for assembly line balancing, since the evaluations are partial, the parameters are not included and the problems used are not standard, therefore the results are not reproducible.

We could state the future research directions in this area as follows:

- To demonstrate the effectiveness of GAs in solving complex manufacturing system design problems, ALBPs can be extended to include features such as parallel workstations, two-sided workstations, U-shaped line layout, mixed model production, assignment restrictions, and stochastic processing times.
- Besides capacity oriented objectives, such as Type-1 and Type-2, the cost or profit oriented objectives can be used to reflect the long-term effects of balancing decisions.
- More effort can be given to study the ALBP as a multi-objective problem.
- In order to improve the performance of a GA, the problem specific knowledge in the form of an additional heuristic optimization algorithm can be included.
- The values of control parameters can be set experimentally over different instances that range in size.
- The comparative studies for performance evaluation can include CPU time as a criterion.
- Standardized, realistic benchmark problems and state-of-the-art solution methods are required for testing and comparing methodical enhancements of GAs.
- A trend in the genetic assembly line balancing practice is to incorporate local search techniques into the main loop of the GA. The main ground for GAs is in areas where improvements can be made by hybridizing methods. Therefore, an interesting future pursuit may be the hybridization of GA.
- A great majority of the researches has been implemented on simple artificial test problems. To draw the attention of practitioners and also to receive more realistic results regarding the performance of the proposed GAs, more effort can be spent on solving real-world complex ALBP using GAs.
- Despite the availability of many effective GAs for different types of ALBPs, their use in practice is limited due to the lack of GA based software. To fill in the perceived gap, the GA based software, which can be easily used by practitioners, can be developed.

With the growth of the published literature, we believe that the use of GAs for solving ALBPs will continue to attract the interest of researchers. We hope that more research involving the implementation on real-world complex line balancing problems will lead to an increased acceptance among the practitioners and thereby, the evolutionary computation methods will be widely accepted as a sound alternative to solve real-life manufacturing optimization problems.

References

Ajenblit, D. A., & Wainwright, R. L. (1998). Applying genetic algorithms to the U-shaped assembly line balancing problem. In *The proceeding of the 1998 IEEE international conference on evolutionary computation* (pp. 96–101). Anchorage, Alaska, USA.

Anderson, E. J., & Ferris, M. C. (1994). Genetic algorithms for combinatorial optimization: The assembly line balancing problem. *ORSA Journal on Computing, 6*, 161–173.

Aytug, H., Khouja, M., & Vergara, F. E. (2003). Use of genetic algorithms to solve production and operations management problems: A review. *International Journal of Production Research, 41*(17), 3955–4009.

Baudin, M. (2002). *Lean assembly: The nuts and bolts of making assembly operations flow*. Productivity, New York.

Bautista, J., Suarez, R., Mateo, M., & Companys, R. (2000). Local search heuristics for the assembly line balancing problem with incompatibilities between tasks. In *The proceedings of the 2000 IEEE international conference on robotics and automation* (pp. 2404–2409). San Francisco, CA.

Baybars, I. (1986a). A survey of exact algorithms for the simple assembly line balancing problem. *Management Science, 32*, 909–932.

Baybars, I. (1986b). An efficient heuristic method for the simple assembly line balancing problem. *International Journal of Production Research, 24*(1), 149–166.

Becker, C., & Scholl, A. (2006). A survey on problems and methods in generalized assembly line balancing. *European Journal of Operational Research, 168*, 694–715.

Bowman, E. H. (1960). Assembly line balancing by linear programming. *Operations Research, 8*(3), 385–389.

Brown, E. C., & Sumichrast, R. T. (2005). Evaluating performance advantages of grouping genetic algorithms. *Engineering Applications of Artificial Intelligence, 18*, 1–12.

Brudaru, O., & Valmar, B. (2004). Genetic algorithm with embryonic chromosomes for assembly line balancing with fuzzy processing times. *The 8th international research/expert conference trends in the development of machinery and associated technology*, TMT 2004, Neum, Bosnia and Herzegovina.

Carnahan, B. J., Norman, B. A., & Redfern, M. S. (2001). Incorporating physical demand criteria into assembly line balancing. *IIE Transactions, 33*, 875–887.

Chan, C. C. K., Hui, P. C. L., Yeung, K. W., & Ng, F. S. F. (1998). Handling the assembly line balancing problem in the clothing industry using a genetic algorithm. *International Journal of Clothing Science and Technology, 10*(1), 21–37.

Chen, R. S., Lu, K. Y., & Yu, S. C. (2002). A hybrid genetic algorithm approach on multi-objective of assembly planning problem. *Engineering Applications of Artificial Intelligence, 15*, 447–457.

Cheng, R., Gen, M., & Tsujimura, Y. (1996). A tutorial survey of job-shop scheduling problems using genetic algorithms: part I representation. *Computers & Industrial Engineering, 30*(4), 983–997.

Cheng, R., Gen, M., & Tsujimura, Y. (1999). A tutorial survey of job-shop scheduling problems using genetic algorithms, part II: Hybrid genetic search strategies. *Computers & Industrial Engineering, 36*(2), 343–364.

Coley, D. (2003). *An introduction to genetic algorithms for scientists and engineers*. Singapore: World Scientific Press.

Dar-El, E. M. (1973). MALB-A heuristic technique for balancing large single-model assembly lines. *AIIE Transactions, 5*(4), 343–356.

Dar-El, E. M., & Rubinovitch, Y. (1979). MUST-A multiple solutions technique for balancing single model assembly lines. *Management Science, 25*, 1105–1114.

Dimopoulos, C., & Zalzala, A. M. S. (2000). Recent developments in evolutionary computation for manufacturing optimisation: Problems, solutions and comparisons. *IEEE Transactions on Evolutionary Computation, 4*(2), 93–113.

Erel, E., & Sarin, S. C. (1998). A survey of the assembly line balancing procedures. *Production Planning and Control, 9*, 414–434.

Falkenauer, E. (1991). A genetic algorithm for grouping. In *The proceedings of the fifth international symposium on applied stochastic models and data analysis*. Granada, Spain.

Falkenauer, E. (1997). A grouping genetic algorithm for line balancing with resource dependent task times. In *The proceedings of the fourth international conference on neural information processing* (pp. 464–468). New Zealand.

Falkenauer, E. (1998). *Genetic algorithms for grouping problems*. New York: Wiley.

Falkenauer, E., & Delchambre, A. (1992). A genetic algorithm for bin packing and line balancing. In *The proceedings of the 1992 IEEE international conference on robotics and automation* (pp. 1189–1192). Nice, France.

Ghosh, S., & Gagnon, R. J. (1989). A comprehensive literature review and analysis of the design, balancing and scheduling of assembly systems. *International Journal of Production Research, 27*, 637–670.

Goldberg, D. E. (1989). *GAs in search, optimization and machine learning*. Reading, Massachusetts: Addison-Wesley.

Goncalves, J. F., & De Almedia, J. R. (2002). A hybrid genetic algorithm for assembly line balancing. *Journal of Heuristic, 8*, 629–642.

Held, M., Karp, R. M., & Shareshian, R. (1963). Assembly line balancing-dynamic programming with precedence constraints. *Operations Research, 11*, 442–459.

Helgerson, N. B., & Birnie, D. P. (1961). Assembly line balancing using the ranked positional weight technique. *Journal of Industrial Engineering, 12*(6), 394–398.

Holland, J. H. (1975). *Adaptation in natural and artificial systems*. Ann Arbor, Michigan: The University of Michigan Press.

Iyer, S. K., & Saxena, B. (2004). Improved genetic algorithm for the permutation flow shop scheduling problem. *Computers & Operations Research, 31*(4), 593–606.

Jackson, J. R. (1956). A computing procedure for a line balancing problem. *Management Science, 2*, 261–272.

Karp, R. M. (1972). Reducibility among combinatorial problems. In R. E. Miller & J. W. Thatcher (Eds.), *Complexity of computer applications* (pp. 85–104). New York: Plenum Press.

Kim, Y. K., Kim, Y. J., & Kim, Y. H. (1996). Genetic algorithms for assembly line balancing with various objectives. *Computers & Industrial Engineering, 30*(3), 397–409.

Kim, Y. J., Kim, Y. K., & Cho, Y. (1998a). A heuristic-based genetic algorithms for workload smoothing in assembly lines. *Computers & Operations Research, 25*(2), 99–111.

Kim, Y. K., Kim, Y., & Lee, T. O. (1998b). Two-sided assembly line balancing models. *Working Paper, Department of Industrial Engineering, Chonnam National University*, Korea.

Kim, Y. K., Kim, Y., & Kim, Y. J. (2000). Two-sided assembly line balancing: A genetic algorithm approach. *Production Planning and Control, 11*(1), 44–53.

Leu, Y. Y., Matheson, L. A., & Rees, L. P. (1994). Assembly line balancing using genetic algorithms with heuristic generated initial populations and multiple criteria. *Decision Sciences, 15*, 581–606.

Levitin, G., Rubinovitz, J., & Shnits, B. (2006). A genetic algorithm for robotic assembly line balancing. *European Journal of Operational Research, 168*, 811–825.

Martens, J. (2004). Two genetic algorithms to solve a layout problem in the fashion industry. *European Journal of Operational Research, 154*(1), 304–322.

Martinez, U., & Duff, W. S. (2004). Heuristic approaches to solve the U-shaped line balancing problem augmented by genetic algorithms. In *The proceedings of the 2004 systems and information engineering design symposium*, pp. 287–293.

Miltenburg, J. (2002). Balancing and sequencing mixed-model U-shaped production lines. *International Journal of Flexible Manufacturing Systems, 14*, 119–151.

Miltenburg, J., & Wijngaard, J. (1994). The U-line line balancing problem. *Management Science, 40*(10), 1378–1388.

Mitchell, M. (1996). *An introduction to genetic algorithms*. Cambridge: The MIT Press.

Noorul Haq, A., Jayaprakash, J., & Rengarajan, K. (2006). A hybrid genetic algorithm approach to mixed-model assembly line balancing. *International Journal of Advanced Manufacturing Technology, 28*, 337–341.

Optiline. www.optimaldesign.com/OptiLine/OptiLine.htm.

Peterson, C. (1993). A tabu search procedure for the simple assembly line balancing problem. In *The proceedings of the decision science institute conference* (pp. 1502–1504). Washington, DC.

Ponnambalam, S. G., Aravindan, P., Naidu, G., & Mogileeswar, G. (2000). Multi-objective genetic algorithm for solving assembly line balancing problem. *International Journal of Advanced Manufacturing Technology, 16*(5), 341–352.

Rekiek, B. (2000). *Assembly line design (multiple objective grouping genetic algorithm and the balancing of mixed-model hybrid assembly line)*. PhD Thesis, Free University of Brussels, CAD/CAM Department, Brussels, Belgium.

Rekiek, B., de Lit, P., Pellichero, F., Falkenauer, E., & Delchambre, A. (1999). Applying the equal piles problem to balance assembly lines. In *The proceedings of the ISATP 1999* (pp. 399–404). Porto, Portugal.

Rekiek, B., Dolgui, A., Delchambre, A., & Bratcu, A. (2002). State of art of optimization methods for assembly line design. *Annual Reviews in Control, 26*, 163–174.

Rubinovitz, J., & Levitin, G. (1995). Genetic algorithm for assembly line balancing. *International Journal of Production Economics, 41*, 343–354.

Sabuncuoglu, I., Erel, E., & Tanyer, M. (2000). Assembly line balancing using genetic algorithms. *Journal of Intelligent Manufacturing, 11*(3), 295–310.

Salveson, M. E. (1955). The assembly line balancing problem. *Journal of Industrial Engineering, 6*, 18–25.

Scholl, A. (1999). *Balancing and sequencing of assembly lines*. Heidelberg: Physica-Verlag.

Scholl, A. & Becker, C. (2006). State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. *European Journal of Operational Research, 168*, 666–693.

Simaria, A. S., & Vilarinho, P. M. (2001a). A genetic algorithm approach for balancing mixed model assembly lines with parallel workstations. In *The proceedings of the 6th annual international conference on industrial engineering theory, applications and practice*, November 18–20, 2001. San Francisco, USA.

Simaria, A. S., & Vilarinho, P. M. (2001b). The simple assembly line balancing problem with parallel workstations-a simulated annealing approach. *International Journal of Industrial Engineering, 8*(3), 230–240.

Simaria, A. S., & Vilarinho, P. M. (2004). A genetic algorithm based approach to mixed model assembly line balancing problem of type II. *Computers and Industrial Engineering, 47*, 391–407.

Stockton, D. J., Quinn, L., & Khalil, R. A. (2004a). Use of genetic algorithms in operations management Part 1: Applications. *Proceeding of the Institution of Mechanical Engineers-Part B: Journal of Engineering Manufacture, 218*(3), 315–327.

Stockton, D. J., Quinn, L., & Khalil, R. A. (2004b). Use of genetic algorithms in operations management Part 2: Results. *Proceeding of the Institution of Mechanical Engineers-Part B: Journal of Engineering Manufacture, 218*(3), 329–343.

Suresh, G., & Sahu, S. (1994). Stochastic assembly line balancing using simulated annealing. *International Journal of Production Research, 32*(8), 1801–1810.

Suresh, G., Vinod, V. V., & Sahu, S. (1996). A genetic algorithm for assembly line balancing. *Production Planning and Control, 7*(1), 38–46.

Talbot, F. B., Patterson, J. H., & Gehrlein, W. V. (1986). A comparative evaluation of heuristic line balancing techniques. *Management Science, 32*, 430–454.

Tsujimura, Y., Gen, M., & Kubota, E. (1995). Solving fuzzy assembly line balancing using genetic algorithms. *Computers & Industrial Engineering, 29*(1–4), 543–547.

Valente, S. A., Lopes, H. S., & Arruda, L. V. R. (2002). Genetic algorithms for the assembly line balancing problem: A real-world automotive application. In R. Roy, M. Köppen, S. Ovaska, T. Fukuhashi, & F. Hoffman (Eds.), *Soft computing in industry - recent applications* (pp. 319–328). Berlin: Springer-Verlag.