Balancing mass production machining lines with genetic algorithms

Olga Guschinskaya[1], Evgeny Gurevsky[1], Anton Eremeev[2], and Alexandre Dolgui[1]

[1] École Nationale Supérieure des Mines de Saint-Étienne
158, cours Fauriel, 42023 Saint-Étienne Cédex 2, France
{guschinskaya, gurevsky, dolgui}@emse.fr
[2] Omsk Branch of Sobolev Institute of Mathematics SB RAS
Pevstsov St. 13, 644099 Omsk, Russia
eremeev@ofim.oscsbras.ru

Abstract. A balancing problem for serial machining lines with multi-spindle heads is studied. The objective is to assign a set of given machining operations to a number of machines while minimizing the line cost and respecting a number of given technological and economical constraints. To solve this problem, three different genetic algorithms are suggested and compared via a series of numerical tests. The results of computational experiments are presented and analyzed.

1 Introduction

Balancing serial machining lines with multi-spindle heads is a very hard combinatorial problem. It consists in assigning a set of given machining operations to a sequence of transfer machines. The number and the configuration of these machines are not known in advance and they must be defined by optimization procedure. In fact, each transfer machine can be equipped with a number of multi-spindle heads activated in sequence. The number and the order of activation of multi-spindle heads depend on the operations assigned to each spindle head. A set of operations assigned to the same spindle head is referred to as block. All operations within a block are executed simultaneously. The parallel execution of operations is possible due to the fact that multi-spindle heads carry several tools activated by the same driver. The assignment of an operation to a spindle head defines which tool must be fixed and in which position. Therefore, the final operations assignment defines the configuration of each spindle head and, as a consequence, each transfer machine. This assignment is subject to numerous technological constraints which must be respected for a feasible machining process. The design objective is to find a configuration of a line which satisfies all given technological and economical constraints and minimizes the line cost that depends directly on the number of used equipments.

Since the presented balancing problem is NP-hard, using optimization tools becomes indispensable for designers who seek for taking efficient decisions. The

first mathematical model for this problem has been presented in [3] and referred to as the Transfer Line Balancing Problem (TLBP). Several exact and heuristic methods have been developed to solve TLBP, a quantitative comparison of the proposed solution methods has been presented in [8]. The numerical tests showed that solving exactly industrial instances of this problem is time consuming and inapplicable for end users who have to deal in practice with designing a number of lines at the same time. As a consequence, powerful metaheuristic approaches are clearly needed for effective solving real-size problem instances. In this paper, three different genetic approaches are suggested for getting relatively good solutions for real case problems.

2 Problem Statement

For a TLBP problem instance, the following input data is assumed to be given:

- $\mathbf{N}$ is the set of all operations involved in machining of a part;
- t_j is the processing time of operation j, $j \in \mathbf{N}$;
- T_0 is the maximal admissible line cycle time;
- τ^S and τ^b are the auxiliary times needed for activation of a machine and a spindle head, respectively;
- C_1 and C_2 are the costs of one machine and one spindle head;
- m_0 is the maximal admissible number of machines;
- n_0 is the maximal number of spindle heads per machine.

The following constraints must be taken into account:

- *Precedence constraints* between the operations. These constraints define non-strict partial order relation over set of operations $\mathbf{N}$. They are represented by a digraph $G = (\mathbf{N}, D)$. An arc $(i, j) \in \mathbf{N}^2$ belongs to set D if and only if the block with operation j cannot precede the block with operation i. If $(i, j) \in D$ then operations i and j can be performed simultaneously in a common block.
- *Inclusion constraints* defining the groups of operations that must be assigned to the same machine, because of a required machining tolerance. These constraints can be represented by a family I^{m} of subsets of $\mathbf{N}$, such that all operations of the same subset $e \in I^{\mathrm{m}}$ must be assigned to the same machine.
- *Exclusion constraints* defining the groups of operations that cannot be assigned to the same machine because of their technological incompatibility. These constraints are represented by a family E^{m} of subsets of $\mathbf{N}$, such that all elements of the same subset $e \in E^{\mathrm{m}}$ cannot be assigned to the same machine.
- *Cycle time constraint:* Let N_k be the set of operations of machine k, $k \in \{1, \ldots, m\}$, $m \leq m_0$, m is the number of machines in a solution S. Let N_{kl} be the set of operations grouped into common block l, $l \in \{1, \ldots, n_k\}$, of machine k, $n_k \leq n_0$, where n_k is the number of blocks of machine k in

solution S. Using these notations, the cycle line constraint can be introduced like follows:

$$\sum_{l=1}^{n_k} t^b(N_{kl}) + \tau^S \leq T_0, \quad k \in \{1, \ldots, m\},$$

where $t^b(N_{kl}) = \max\{t_j : j \in N_{kl}\} + \tau^b$ is the time of l-th block of k-th machine.

Therefore, a solution S of TLBP can be represented by a collection $S = \{\{N_{11}, \ldots, N_{1n_1}\}, \ldots, \{N_{m1}, \ldots, N_{mn_m}\}\}$, determining an assignment of $\mathbf{N}$ to machines and blocks. Solution S is feasible, if it satisfies all constraints described above. The studied problem consists in minimizing the investment costs which can be represented as follows:

$$C(S) = C_1 m + C_2 \sum_{k=1}^{m} n_k \rightarrow \min_{S \in \mathbf{S}},$$

where $\mathbf{S}$ is the set of all feasible solutions.

3 Genetic Algorithm: General Approach

A genetic algorithm (GA) is a metaheuristic which uses some mechanisms inspired by biological evolution: reproduction, mutation, crossover, and selection see e.g. [9, 10]. Such an approach has been already successfully applied for solving different balancing problems, see for instance the review [11].

A genetic algorithm starts with an initial population which contains a number of individuals. In our implementation, each individual represents a feasible solution for the studied problem. These individuals are generated by using a heuristic. Each individual is characterized by the fitness function (the cost of the solution in our case) determining the chances of its survival.

The encodings of individuals are usually called genotypes. In this paper, three different techniques of encoding are presented and compared on a series of benchmark problems.

To model the evolution of a population, two parents (individuals) are chosen from the population and are used to create an offspring (new individual). In the presented implementation of the GAs, each parent is selected using the s-tournament method: s individuals are taken at random from the current population and the one with the best fitness is selected as a parent. Therefore, the population size N_{size_pop} remains constant during the execution of a GA.

New individuals are built by means of a reproduction operator that usually consists of crossover and mutation procedures. The crossover procedure produces an offspring from two parent individuals by combining and exchanging their elements. The mutation procedure adds small random changes to an individual. Finally, the obtained offspring replaces the worst individual in the current population, if the offspring has better value of the fitness function.

The algorithm is restarted with a new initial population every time the number of iterations during which the current solution has not been improved is equal to I_{non_imp}. This continues until the total execution time reaches the limit T_{max}. The best solution found over all runs is returned as the final output.

The aim of this paper is to find the most efficient genetic algorithm for balancing real-size machining lines with multi-spindle heads. To do it, three different GAs are presented and compared. The first one is based on MIP formulation of TLBP and, as a consequence, uses a binary-based encoding of individuals. This algorithm is described in Section 4. The second and the third genetic algorithms presented in Section 5 and 6, respectively, employ other mutation and crossover procedures and solution encodings based on using different heuristics. The performances of these algorithms are compared in Section 7 on a series of benchmark problems.

4 GA1: Binary-Based Encoding

The MIP model of TLBP proposed in [3] is employed by this algorithm. The individuals are coded using binary variables X_{jq}, $j \in \mathbf{N}$, $q \leq n_0 m_0$: X_{jq} equals 1 if operation j is assigned to block q, 0 otherwise. These variables are used to describe the operations assignment to blocks in a feasible solution.

In this genetic algorithm, the initial population of individuals is obtained by heuristic suggested in [4]. Each individual is represented by a sequence of values of variables X_{jq}. A MIP-recombination operator is used to obtain an offspring from the selected parents. Firstly this operator fixes all Boolean variables equal to their values in parent p_1 and then it releases (with probability P_c) all Boolean variables having different values in p_1 and p_2 (analogue of crossover). Finally, it changes (with probability P_m) the values of randomly selected variables (analogue of mutation). The obtained MIP problem (with some fixed variables) is solved by a MIP solver.

In our implementation of GA1, ILOG CPLEX 9.0 is used to solve MIP problems. To avoid time-consuming computations, a limit time T_{rec} is fixed to the solver at each call. If the solver does not return a feasible solution to the subproblem, then the MIP-recombination outputs a genotype obtained by the standard mutation procedure.

The second and the third genetic algorithms (GA2 and GA3, respectively) use a non-binary based encoding of the individuals, like in [2]. This means that the way to obtain a feasible solution and not the solution itself is encoded. In the presented algorithms, two different heuristic methods are used for constructing feasible solutions from individuals. The two algorithms are based on using two different heuristics.

5 GA2: Encoding Based on GBL Heuristic

GA2 uses GBL (Greedy Blocks Loading) heuristic [6] for solution encoding. An individual consists of a sequence of parameters that are used by GBL heuristic

for constructing a feasible solution. Here, a brief description of this heuristic is given, for more details see [6].

The solution construction starts with one machine with an empty block. Then, operations are assigned to the current machine by adding new blocks. When no more blocks can be created at the current machine, a new machine is created. This continues until all operations are assigned. To select an operation to be assigned to the current machine, list CL (Candidate List) is used. This list contains all operations that can be assigned in the current moment, i.e. for which precedence constraints are satisfied and there is no exclusion constraint with the operations assigned to the current block and machine. When CL is constructed, a greedy function is applied to each candidate operation and the operations are ranked according to their greedy function values. Well ranked candidate operations are placed into a restricted candidate list (RCL). Then, an element is randomly selected from RCL and added to the solution. Taking this into account, list CL is updated.

In GA2, instead of one determined greedy function, different criteria are imployed for ranking operations in list CL. The following greedy functions can be used:

- Lower bound on the number of blocks required to assign all successors of operation i with condition that i will be assigned to the current block;
- Lower bound on the number of blocks required to assign immediate successors of operation i with condition that i will be assigned to the current block;
- Total number of all successors of operation i ;
- Total number of immediate successors of operation i ;
- Processing time of operation i.

Two possible techniques for constructing list RCL are also considered:

$$RCL = \{j \in CL : g(j) \geq g_{max} - \alpha(g_{max} - g_{min})\},$$

in order to select the operations with greater greedy values, and

$$RCL = \{j \in CL : g(j) \leq g_{min} - \alpha(g_{min} - g_{max})\},$$

to select the operations with smaller greedy values. Here g_{max} and g_{min} are respectively the minimum and the maximum of greedy function $g(j)$ over list CL. Parameter $\alpha \in [0, 1]$ controls the trade-off between randomness and greediness in the construction process.

Therefore, each time list RCL is built, one of ten (2*5) criteria can be used. Taking into account the fact that each operation is to be chosen one time, an individual is represented by the number of genes equal to the number of operations. The value of each gene is the code of the criterion to be used for constructing list RCL. The first time, RCL is build using the criterion corresponding to the value of the first gene and so on. The initial population is created by attributing random values to genes.

The fitness value of an individual is measured by the cost of the corresponding solution obtained by GBL heuristic.

The crossover procedure is implemented as follows: each child's gene inherits the value of the corresponding gene either from parent p_1 (with probability P_c) or from parent p_2 (with probability $1 - P_c$). The mutation procedure selects at random two child's genes and increases (with probability P_m) or decreases (with probability $1 - P_m$) by one the integer value for any gene between these two positions.

A local search procedure presented in [6] is applied to the best individual in the current population. The used method is based on the decomposition and aggregate solving of sub-problems by a graph approach.

6 GA3: Encoding Based on FSIC Heuristic

This genetic algorithm (GA3) uses FSIC (*First Satisfy Inclusion Constraints*) heuristic for constructing feasible solutions from individuals. The last version of this heuristic and its parameters have been presented in [7]. Therefore, the encoding of individuals is based on the use of the parameters that the heuristic employs during the construction of a feasible solution for a TLBP problem.

In heuristic FSIC, as in heuristic GBL, the operations are assigned one by one and new blocks and machines are created when necessary. List CL is also used, but list RCL is not employed. As it was presented in [7], the selection of the operation to be assigned depends on 4 control parameters: $check\_time \in \{0, 1\}$, $check\_blocks \in \{0, 1, 2\}$, $divide\_L_2 \in \{0, 1\}$, $trials\_L_2 \in \{0, 1\}$. Their values can be changed each time list CL is constructed. The results of test experiments presented in [7] have been used for selecting the ten most effective combinations of these parameters. A code was assigned to each of these ten combinations. These codes are used for individuals encoding in GA3. Taking into account the fact that each operation is to be chosen once, an individual is represented by the number of genes equal to the number of operations. The value of each gene is the code of a combination of the control parameters. The solution construction starts with the parameters corresponding to the value of the first gene. Then, each time list CL is rebuilt, the values of the control parameters are changed accordingly to the value of next gene. The fitness value of an individual is measured by the cost of the corresponding solution obtained by FSIC heuristic.

The initial population is generated randomly. The crossover and mutation procedures are the same as in GA2. As for GA2, a local search procedure presented in [6] is applied to the best individual in the current population.

7 Experimental Results

The purpose of this section is to compare the performances of three proposed genetic algorithms GA1, GA2, GA3 and the best heuristic method FSIC (the last version of this heuristic can be found in [7]). To do this, 2 series of randomly

generated (S1-S2) and 2 series (SI1-SI2) of industrial instances have been used. All experiments were carried out on Pentium-IV (3 GHz, 1.5 RAM).

For series S1-S2, the following problem parameters are used: $C_1 = 10$, $C_2 = 2$, $T_0 = 100$, $\tau^b = \tau^S = 0$, $n_0 = 4$. Each series of S1-S2 contains 50 test problems. The number of operations and the order strength ($OS = \frac{2|D|}{|\mathbf{N}|(|\mathbf{N}|-\mathbf{1})}$) are as follows: $|\mathbf{N}| = 25$ and $OS = 0.5$ for Series S1, $|\mathbf{N}| = 50$ and $OS = 0.9$ for Series S2. The available time per test was limited by 90 sec for S1 and by 300 sec for S2.

The following notations are used for the presentation of the obtained results: *NO* is the number of instances where the optimal solutions were obtained; Δ_{max}, Δ_{av} are respectively the percentage of maximal and average deviation of a obtained solution from the optimal one for the same problem instance.

Table 1. Results for S1-S2

	S1				S2			
	GA1	GA2	GA3	FSIC	GA1	GA2	GA3	FSIC
NO	**50**	**50**	**50**	23	49	**50**	49	18
Δ_{max}, %	**0**	**0**	**0**	10.53	1.72	**0**	2.13	6.00
Δ_{av}, %	**0**	**0**	**0**	3.15	0.03	**0**	0.04	2.14

The results presented in Table 1 show that for Series S1-S2 the proposed genetic algorithms demonstrated rather similar performances. Algorithm GA2 found 100% of optimal solutions, GA1 and GA3 reached 99% of optimums while heuristic FSIC found only 41% of optimal solutions.

The both Series SI1-SI2 contain 20 industrial problems. The following parameters are the same for all these problem instances: $C_1 = 1$, $C_2 = 0.5$, $T_0 = 5.15$, $\tau^b = 0.2$, $\tau^S = 0.4$, $n_0 = 4$. The maximal, average and minimal numbers of operations are respectively 92, 72, 46 for the instances from Series SI1 and 127, 108, 87 for the instances from Series SI2. The maximal, average and minimal values of the order strength are respectively 0.53, 0.45, 0.39 for the instances from Series SI1 and 0.5, 0.43, 0.33 for the instances from Series SI2. The available solution time was limited by 720 sec for SI1 and 2400 sec for SI2.

Taking into account the fact that for these test series the optimal solutions are not known yet, the solutions obtained by GAs and FSIC are compared with the best known solutions. The following notations are used for the presentation of the obtained results: *NB* is the number of instances where the best known solution was obtained; Δ_{max}, Δ_{av}, Δ_{min} are the percentage of maximal, average and minimal deviation of a obtained solution from the best known one.

With respect to the results given in Table 2 GA2 provided in average a better solution than GA1, GA3 and FSIC. Algorithm GA2 found 38 best known solutions for 40 test problems while GA1, GA3 and FSIC obtained only 7, 3 and 1 ones, respectively. The average deviation of GA2 for series SI1-SI2 is equal to 0.51% while it is 2.79%, 4.32% and 18.92% for GA1, GA3 and FSIC, respectively.

Table 2. Results for SI1-SI2

	SI1				SI2			
	GA1	GA2	GA3	FSIC	GA1	GA2	GA3	FSIC
NB	3	**20**	3	1	4	**18**	0	0
Δ_{max}, %	7.14	**0**	10	32	7.04	**0.09**	10.84	45.07
Δ_{av}, %	3.34	**0**	3.71	13.33	2.23	**1.01**	4.93	24.50
Δ_{min}, %	**0**	**0**	**0**	**0**	**0**	**0**	0.78	13.85

Therefore, it can be concluded that all suggested methods outperformed heuristic FSIC and algorithm GA2 performs in average better than other methods.

References

1. Baybars, I.: A survey of exact algorithms for the simple assembly line balancing. Management Science. 32, 909–932 (1986)
2. Baykasoğlu, A., Özbakir, L.: Stochastic U-line balancing using genetic algorithms. The International Journal of Advanced Manufacturing Technology. 32, 139–147 (2007)
3. Dolgui, A., Finel, B., Guschinsky, N.N., Levin, G.M., Vernadat, F.B.: MIP approach to balancing transfer lines with blocks of parallel operations. IIE Transactions. 38, 869–882 (2006)
4. Dolgui, A., Guschinskaya, O., Eremeev, A.: MIP-based GRASP and genetic algorithm for balancing transfer lines. Matheuristics, Annals of Information Systems, V. Maniezzo, T. Stützle, S. Voß (Eds.), Springer US. 10, 189–208 (2010)
5. Ghosh, S., Gagnon, R.: A comprehensive literature review and analysis of the design, balancing and scheduling of assembly systems. International Journal of Production Research. 27, 637–670 (1989)
6. Guschinskaya, O., Dolgui, A.: Équilibrage des lignes d'usinage à boîtiers multi-broches avec la méthode GRASP. Actes de la 7ème Conférence Internationale de Modélisation et Simulation (MOSIM'08). 2, 1121–1130 (2008)
7. Guschinskaya, O., Dolgui, A.: A transfer line balancing problem by heuristic methods: industrial case studies. Decision Making in Manufacturing and Services. 2, 33–46 (2008)
8. Guschinskaya, O., Dolgui, A.: Comparison of exact and heuristic methods for a transfer line balancing problem. International Journal of Production Economics. 120, 276–286 (2009)
9. Holland, J.H.: Adaptation in natural and artificial systems. University of Michigan Press (1975)
10. Reeves, C.R.: Feature article – genetic algorithms for the operations researcher. INFORMS Journal on Computing. 9, 231–250 (1997)
11. Scholl, A., Becker, C.: State-of-the-art exact and heuristic solution procedures for the simple assembly line balancing. European Journal of Operational Research. 168, 666–693 (2006)