Int J Adv Manuf Technol (2011) 55:11–22
DOI 10.1007/s00170-010-3053-0

ORIGINAL ARTICLE

Metaheuristic approaches for the design of machining lines

Olga Guschinskaya · Evgeny Gurevsky · Alexandre Dolgui · Anton Eremeev

Received: 5 September 2010 / Accepted: 18 November 2010 / Published online: 4 December 2010

Abstract The considered optimization problem deals with the design of the machine equipment used in a serial paced line. The cost and effectiveness of such a line depend on this decision. The investment cost can be reduced by optimizing the assignment of machining operations to the pieces of equipment. In this paper, new powerful metaheuristic methods applying the principles of a greedy randomized adaptive search procedure and a genetic algorithm are suggested for this problem. These methods are evaluated on a series of benchmarks and real industrial problems.

Keywords Line design · Machining lines · GRASP · Genetic algorithms

1 Introduction

The studied problem consists in optimizing the design of unit-built machines used for the production of a given part. Such machines are arranged in a serial paced line. "Serial" means that the part visits all machines in the order of their installation. "Paced" means that the time spent by the part at each machine is limited by the line cycle time, denoted by T_0. Machines are called "unit-built" since they are assembled with independently constructed units (multi-spindle heads). When a part arrives to a machine, the first multi-spindle head installed there is activated followed by the second one and so on. Figure 1 presents a scheme of such machining lines.

Each multi-spindle head carries several machining tools which are applied to the part simultaneously to execute a set of machining operations in parallel. The content of each unit must be obtained from the known set of operations required for completing a given part. For this purpose, the operations must be grouped in "blocks". Each block corresponds to a multi-spindle head and defines the set of operations to be executed in parallel. Then, the set and the activation order of units at each machine must be determined.

While grouping operations together, their technological compatibility has to be checked. Moreover, the precedence, the capacity, and the cycle time constraints must be taken into account. These constraints and other input data are detailed in Section 2. The objective is to find an assignment of operations to blocks and the blocks to machines which satisfies all given constraints and minimizes the total investment cost related to machines and multi-spindle heads.

This design problem is combinatorially hard and, as a consequence, using optimization tools becomes indispensable for designers who seek for taking efficient solutions in short amount of time. The first mathematical model taking into account the most of technological parameters and constraints of this design problem has been presented in [4], where a mixed integer program has also been developed. Recently, some other exact and simple heuristic methods have been proposed for this problem; their comparison can be found in [10].

O. Guschinskaya (✉) · E. Gurevsky · A. Dolgui
École des Mines de Saint-Étienne, Saint-Étienne, France
e-mail: guschinskaya@emse.fr

A. Eremeev
Omsk Branch of Sobolev Institute of Mathematics SB RAS, Omsk, Russia

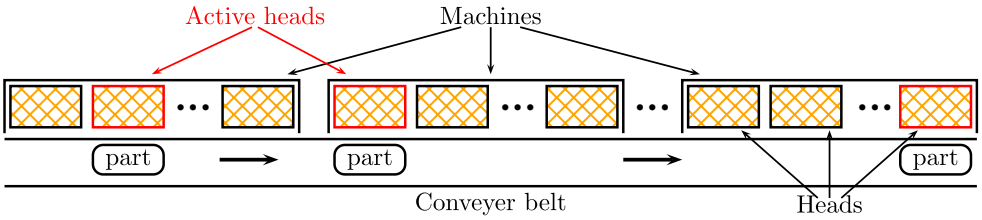

Fig. 1 Serial paced lines equipped with unit-built machines

Because of their combinatorial complexity, solving industrial instances of this problem exactly is time consuming [10]. Thus, exact methods are inapplicable for end users. As a consequence, powerful metaheuristic approaches are clearly needed to tackle real-size problem instances. The first step in this direction has been made in [3] where two metaheuristic methods have been combined with a mixed integer program. In this paper, two new approaches are presented in Sections 3 and 4. The results of their evaluation on a series of benchmarks and real industrial problems as well as the comparison with the results obtained in [3] are reported in Section 5. Concluding remarks are given in Section 6.

2 Problem statement

The given set of operations is denoted by **N**. The number of machines m and the number of multi-spindle heads n_k at kth machine ($k = 1, 2, ..., m$) to which the operations have to be assigned are not fixed a priori, but they are limited by the space and technological constraints. The number of machines must not exceed m_0 ($m \leq m_0$) and each machine can be equipped with at maximum n_0 multi-spindle heads ($n_k \leq n_0$, $k = 1, 2, ..., m$). The installation of a machine implies a cost estimated by C_1 units. The fabrication cost of a multi-spindle head is estimated by C_2 units.

Each operation $j \in \mathbf{N}$ is given by its machining time t_j or, alternatively, it may be characterized by two parameters: the required working stroke length λ_j and the maximal admissible feed per minute s_j. The working stroke length includes the required depth of cut and the distance between the tool and the part surface as it is shown in Fig. 2. The feed per minute is a measure for the cutting speed.

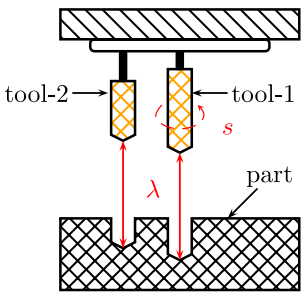

Fig. 2 A multi-spindle head equipped with 2 tools

Let N_k be the set of all operations assigned to machine k and N_{kl} be the set of operations grouped into block l of machine k. All unproductive time needed to launch and stop a multi-spindle head is integrated in τ^{b}. All unproductive time related to loading and unloading a part on a machine is denoted by τ^{m}.

Two manners of calculating the time required for processing block of operations N_{kl} are used in industry. The first one uses the assumption that the block processing time $t^{\mathrm{b}}(N_{kl})$ depends on the longest operation in the block:

$$t^{\mathrm{b}}(N_{kl}) = \max\{\, t_j \mid j \in N_{kl} \,\} + \tau^{\mathrm{b}}. \tag{1}$$

The second one uses the parameters λ_j and s_j:

$$t^{\mathrm{b}}(N_{kl}) = \frac{\max\{\, \lambda_j \mid j \in N_{kl} \,\}}{\min\{\, s_j \mid j \in N_{kl} \,\}} + \tau^{\mathrm{b}}. \tag{2}$$

The latter definition allows calculating the machining time more accurately, but introducing the machining parameters for operations increases dramatically the complexity of the optimization problem to solve. Note that the latter definition covers the first case, assuming $\lambda_j = t_j$, $s_j = 1$ for all j.

The machine processing time $t^{\mathrm{m}}(N_k)$ is calculated as the sum of its block processing times, since the multi-spindle heads of a machine are activated sequentially. For the paced lines considered here, the time that a part spends on a machine is limited by the line cycle. This constraint is called the *cycle time constraint* and can be expressed as follows:

$$t^{\mathrm{m}}(N_k) = \sum_{l=1}^{n_k} t^{\mathrm{b}}(N_{kl}) + \tau^{\mathrm{m}} \leq T_0.$$

The repartition of the operations must also respect the assignment constraints related to the technological process:

- *Precedence constraints*. They define non-strict partial order relations among operations and are given by a digraph $G = (\mathbf{N}, D)$. An arc $(i, j) \in \mathbf{N}^2$ belongs

to set D if and only if operation i must be executed before or at the same time as operation j. Such operations i and j can be performed simultaneously by the same multi-spindle head using special tools.

- *Inclusion constraints*. They impose to assign certain groups of operations to the same machine because of a required machining tolerance. These constraints are 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.

- *Machine and Block exclusion constraints*. They verify that the groups of operations which are technologically incompatible (at machine or block level) are not assigned to the same machine (respectively block). These constraints are represented by a family E^{m} (respectively E^{b}) of subsets of $\mathbf{N}$, such that all elements of the same subset $e \in E^{\mathrm{m}}$ (respectively $e \in E^{\mathrm{b}}$) cannot be assigned to the same machine (respectively block). However, any proper subset $e^* \subset e$ of operations can be assigned to the same machine (respectively block).

The goal is to minimize the investment costs required for the installation of the line being designed. Using the introduced notations, the objective function can be expressed as follows:

$$\text{Min } C_1 m + C_2 \sum_{k=1}^{m} n_k.$$

In this article, two new metaheuristic methods are suggested for solving this optimization problem. Both manners of calculating the block processing time can be easily integrated in them. The presentation starts in Section 3 with an approach based on greedy randomized adaptive search procedure (GRASP).

3 GRASP-based approach

3.1 General scheme of the approach

GRASP is a metaheuristic which can be used for solving a large number of combinatorial problems. For example, this method has been already successfully applied for the assignment of operations in assembly lines with sequence-dependent setup times [1, 18].

GRASP is usually implemented as a multi-start procedure. Each of its iteration consists of several phases, where feasible solutions are obtained by a semi-greedy heuristic and then improved by a local search [7, 13]. Such phases are repeated interchangeably until a stopping criterion is satisfied. Extensive bibliographies on GRASP can be found in [8, 22]. The GRASP-based approach suggested here contains three principal phases.

1. The first one, called *construction phase*, aims in obtaining a set of feasible solutions, called *elite set ES*. The first $|ES|$ solutions found using semi-greedy heuristic greedy blocks loading (GBL; described in Section 3.2) constitute this set.
2. The second phase, called *path-relinking phase*, applies a path-relinking procedure, initially proposed in [9] for Tabu Search method and firstly applied to GRASP in [16]. It consists in generating new feasible solutions, called *intermediate solutions*, by exploring the trajectories that connect high-quality solutions. This procedure is detailed in Section 3.3. If the best intermediate solution has a smaller cost than the worst solution in *ES*, it replaces the latter one in *ES*. After $I_{\mathrm{nimp\_max}}$ iterations of the path-relinking procedure without improving the best solution of a current elite set, the algorithm returns to the construction phase where set *ES* is reinitialized.
3. When the time allocated for the two previous phases is over, the best found solution is treated in the third phase, called *improvement phase*, which aims in improving this solution with a decomposition method presented in Section 3.4.

This continues until the total execution time reaches the allocated time limit.

The three principal phases of the suggested approach are detailed in the next subsections.

3.2 Construction phase

The semi-greedy heuristic developed for the suggested GRASP-based approach is called GBL. This heuristic constructs a feasible solution by assigning as many operations as possible to the current block. At the beginning, the current solution contains only one machine with one empty block. The algorithm assigns operations to this block until no operation can be added because of the exclusion or precedence constraints. Then, if the cycle time constraint is respected, a new block is opened at the current machine, otherwise, a new machine with one empty block is created. When a new block is opened, it becomes current and the algorithm assigns operations to it.

Let m be the index of the current machine and n_m be the index of the current block of machine m. To choose an operation to be assigned to block n_m, the

so-called candidate list (CL) is constructed of unassigned operations. Then, a greedy function is used to evaluate the operations included in list CL. The operations with the best values are selected for a restricted candidate list (RCL). An operation is chosen at random from this list. It is assigned to block n_m. Then, list CL is rebuilt. This continues until either all operations are assigned and a feasible solution is obtained or it is impossible to create a new machine without exceeding m_0. In the last case, the iteration is considered as infeasible.

The algorithms used for constructing the employed lists are detailed here below.

3.2.1 Construction of list CL

List CL contains all operations that can be assigned to block n_m. This list is built in the following way: set of unassigned operations is looked through and operation j is added to CL if all the following conditions are satisfied:

- all predecessors of j have been already assigned;
- assigning j to the current block does not violate the cycle time constraint (the block time is calculated using (1) or (2));
- its assignment to the current machine respects the machine exclusion constraints taking into account the operations already assigned to m, i.e., $\forall e \in E^{\mathrm{m}}$ such that $j \in e$: $e \cap (N_m \cup \{j\}) \neq e$; and
- its assignment to the current block does not violate the block exclusion constraints taking into account the operations already assigned to the current block, i.e., $\forall e \in E^{\mathrm{b}}$ such that $j \in e$: $e \cap (N_{mn_m} \cup \{j\}) \neq e$.

If CL $= \emptyset$, no more operations can be assigned to the current block. A new block is created at the current machine if it is possible, otherwise a new machine is opened and CL is rebuilt. If CL $\neq \emptyset$, then it is used for constructing a Restricted Candidate List discussed in the next subsection.

3.2.2 Construction of list RCL

Different schemes of constructing RCL from CL have been suggested in the literature, see e.g., [7, 13, 21]. Generally, a greedy function is applied to the operations from CL and those with the best values are selected to be included in RCL. In order to diversify the solutions obtained by GBL heuristic, the greedy function is not unique in the suggested algorithm, but there exists the following list of employed greedy functions:

1. Lower bound on the number of blocks required to assign all successors of operation j with the condition that j will be assigned to the current block;
2. Lower bound on the number of blocks required to assign all immediate successors of operation j with the condition that j will be assigned to the current block;
3. Total number of all successors of operation j;
4. Total number of all immediate successors of j;
5. Processing time of j (calculated as λ_j/s_j if operations are given by their parameters); and
6. The number of subsets of the block and machine exclusion constraints where operation j is included.

Moreover, each function can be used in two different manners either

$$\mathrm{RCL}' = \{j \in \mathrm{CL} : g(j) \geq g_{\max} - \alpha(g_{\max} - g_{\min})\}, \quad (3)$$

in order to select the operations with greater values of the greedy function, or

$$\mathrm{RCL}'' = \{j \in \mathrm{CL} : g(j) \leq g_{\min} - \alpha(g_{\min} - g_{\max})\}, \quad (4)$$

to select the operations with smaller values, where $g_{\max}$ and $g_{\min}$ are respectively the minimum and the maximum of greedy function $g(j)$ for the operations in CL.

Parameter $\alpha \in [0, 1]$ controls the trade-off between the randomness and greediness in the construction process. If $\alpha = 0$, then the semi-greedy construction reduces to a greedy algorithm (i.e., at each iteration the best-ranked element is chosen), and if $\alpha = 1$, then the constructive process becomes completely random. Thus, the value of α is used to control the size of list RCL obtained from list CL. In the suggested algorithm, a mechanism of self-adjusting parameter α is used. It is detailed in Appendix A.

Taking into account six available greedy functions and two ways to use each one of them, there are 12 possible manners to construct list RCL. They can be enumerated from 1 to 12 in the following way: number 1 corresponds to the first greedy function used with Formula (3), number 2 relates to the first greedy function used with Formula (4), number 3 links to the second greedy function used with Formula (3) and so on.

By applying a different method each time list RCL is built, more dissimilar solutions can be provided by GBL heuristic. To define which method is used when, a so-called RCL sequence is employed. Actually, it is an array containing $|\mathbf{N}|$ cells, since for assigning $|\mathbf{N}|$ operations, list RCL is built at most $|\mathbf{N}|$ times. A RCL sequence is filled in at random with natural numbers

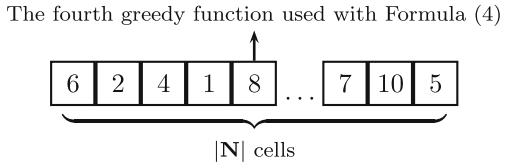

Fig. 3 A RCL sequence

between 1 and 12 before GBL heuristic starts a new solution construction. The number of the method to use when list RCL is built for ith time is stored in ith cell. An example of a RCL sequence is given in Fig. 3, where method number 8, for example, is used when list RCL is built for the 5th time.

The operation to be assigned to the current block is selected at random from list RCL. This operation may participate in inclusion constraints. If these constraints are ignored during the assignment procedure, then a great number of infeasible solutions would be generated. To avoid this, the inclusion constraints have to be checked for the operations chosen to be assigned. For this purpose, the algorithm uses an additional list AL described in the next subsection.

3.2.3 Construction of list AL

The inclusion constraints are analyzed for operation j selected from list RCL to be assigned to the current block. If $j \notin e, \forall e \in I^{\mathrm{m}}$, then AL $= \{j\}$, otherwise, list AL groups all the operations that must be assigned to the same machine as operation j, i.e., the list is made of operation j and all the operations from the set $e \in I^{\mathrm{m}}$ including operation j and all their non-assigned predecessors. It cannot be calculated a priori how many blocks will be needed to assign all $i \in$ AL and, as a consequence, it is impossible to know if it will be possible to assign all $i \in$ AL to the current machine. Taking into account the possibility of a failure while assigning $i \in$ AL, the current state of S_{cur} is stored: $S_{\mathrm{copy}} = S_{\mathrm{cur}}$.

The algorithm tries to assign all the operations from list AL to the current machine. If all the operations from AL have been assigned, CL is rebuilt, otherwise the algorithm restores the state of the current solution before any operation belonging to AL have been assigned: $S_{\mathrm{cur}} = S_{\mathrm{copy}}$. Operation j is deleted from RCL and from CL and another operation is randomly chosen from RCL. If RCL $= \emptyset$ and CL $\neq \emptyset$, then RCL is rebuilt. If CL $= \emptyset$, then no more operation can be assigned to the current block. A new block is created if it is possible, otherwise a new machine is opened.

The first $|ES|$ solutions obtained with heuristic GBL constitute the set of elite solutions. This set is treated in path-relinking phase described here below.

3.3 Path-relinking phase

The path-relinking procedure is implemented as follows. At first, a solution S^* is chosen from ES. Its RCL sequence is used as *guiding solution*. At second, a new RCL sequence is randomly generated and becomes *initiating solution* S^0 for the path relinking. The number of intermediate solutions from S^* to S^0 in set IS is calculated on the basis of the Chebychev distance $d(S^0, S^*)$ between RCL sequences of S^0 and S^* solutions:

$$|IS| = d(S^0, S^*) - 1 = \max\{|\mathrm{RCL}^{S^0}[i] - \mathrm{RCL}^{S^*}[i]| : i \in \mathbf{N}\} - 1,$$

where $\mathrm{RCL}^{S^0}[i]$ and $\mathrm{RCL}^{S^*}[i]$ are the ith cells of the RCL sequences corresponding to the initiating and guiding solutions, respectively. If $\mathrm{RCL}^{S^0}[i]$ and $\mathrm{RCL}^{S^*}[i]$ are quite similar, i.e., $d(S^0, S^*) \in \{0, 1\}$ then new S^0 and S^* are chosen, otherwise the moving direction from S^0 to S^* is determined for each cell i in the following way:

$$\delta[i] = \begin{cases} -1, & \text{if } \mathrm{RCL}^{S^0}[i] - \mathrm{RCL}^{S^*}[i] \text{ is positive,} \\ 1, & \text{if } \mathrm{RCL}^{S^0}[i] - \mathrm{RCL}^{S^*}[i] \text{ is negative,} \\ 0 & \text{otherwise,} \end{cases}$$

where δ is an array with $|\mathbf{N}|$ cells such that $\delta[i] \in \{0, -1, 1\}$.

The intermediate solutions (more precisely their RCL sequences) are constructed on the basis of δ. Each

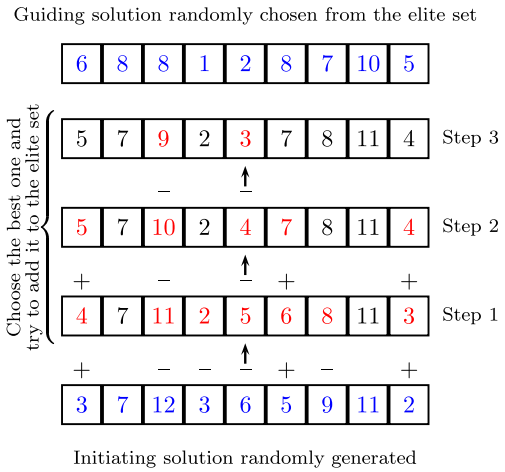

Fig. 4 Path-relinking mechanism

new solution S' is obtained from the current solution S_{cur} as follows: for $\forall i \in \mathbf{N}$: if $|\text{RCL}^{S_{\text{cur}}}[i] - \text{RCL}^{S^*}[i]| \notin \{0, 1\}$, then $\text{RCL}^{S'}[i] \leftarrow \text{RCL}^{S_{\text{cur}}}[i] + \delta[i]$. Then, $S_{\text{cur}} \leftarrow S'$.

An example of the construction of the set of intermediate solutions is given in Fig. 4. Each new solution is obtained from the previous one by modifying its cells by the way defined in array δ. The last intermediate solution is not equal to guiding one, but is close to it.

The cost of each obtained intermediate solution is calculated using GBL heuristic. The RCL sequence of the best intermediate solution replaces the RCL sequence corresponding to the worst solution in *ES*, if the intermediate solution has a better cost value. Therefore, the size of the elite set remains constant during the execution of the PR phase. The improvement phase is applied to the best solution found during each $T_{PR\_\max}$ time-period.

3.4 Improvement phase

The improvement phase starts with a feasible solution obtained at the previous steps and tries to improve it. For this purpose, a decomposition algorithm with aggregate solving of subproblems is used. This method has been already applied with a simple priority heuristic, which constructs a feasible solution without applying any greedy function, in [11]. The general scheme of this method is given in Fig. 5.

The decomposition consists in cutting up the sequence of machines corresponding to a feasible solution into several non intersecting subsequences. The size of each subsequence r is chosen at random, but is limited by two following parameters: $k_{\max}$ which is the maximal number of machines and $N_{\max}$ which is the maximal number of operations per subsequence. The value k_r (the number of machines in subsequence r) is chosen uniformly at random within $[1, k_{\max}]$ and then can be modified so that the total number of operations in a subsequence does not exceed $N_{\max}$. Therefore, the total number of subsequences ω is known only at the end of the decomposition.

Each subsequence r is used to formulate a subproblem SP_r which is of the same type as the initial problem but has a smaller size. It is solved by an exact method. Here, the graph approach described in [5] has been used. When an optimal solution $S(\text{SP}_r)$ is obtained for subproblem SP_r, all blocks of this solution is replaced by macro-operations. They are included in the consecutive subproblem SP_{r+1}. Thus, new operations can be added at step $r + 1$ to the blocks created at steps $1, ..., r$.

As it can be seen, the algorithm constructs a final solution by increasing progressively the size of a subproblem to solve. However, a problem of the initial size is never treated, since the number of blocks belonging to $S(\text{SP}_r)$ is significantly inferior to the total number of operations. The solution of the last subproblem is

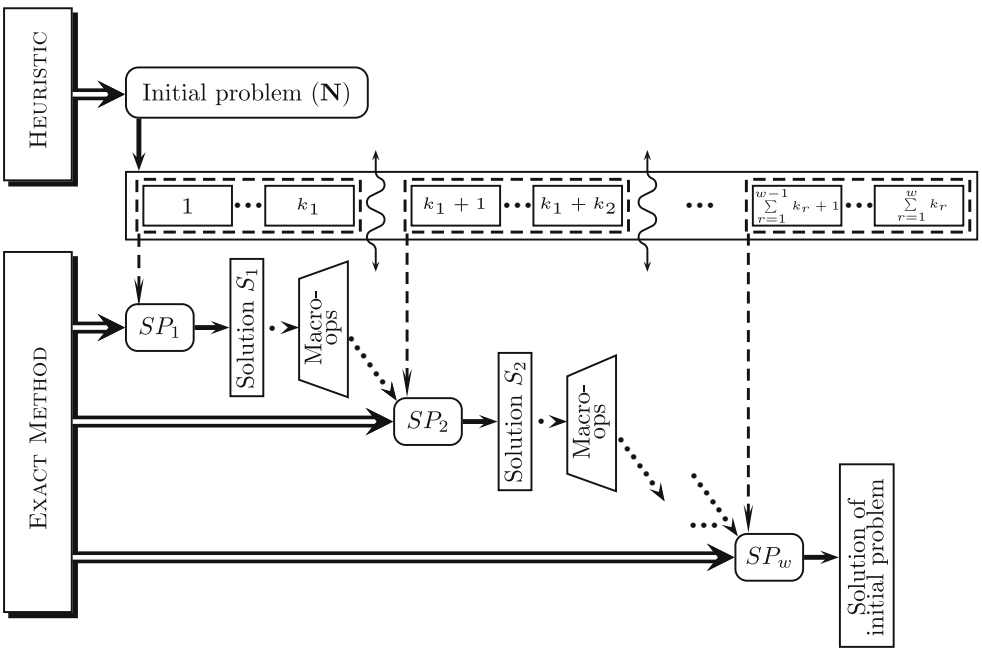

Fig. 5 Scheme of decomposition algorithm with aggregate solving of subproblems

a feasible solution of the initial problem with the cost C_{imp} which is not greater than C_{cur} (the cost of a feasible solution before the decomposition), but not necessarily optimal. Note that the history of creating macro-operations is stored in order to obtain a final solution without macro-operations from a final solution with macro-operations.

4 GA-based approach

4.1 General scheme of the approach

A genetic algorithm (GA) is a metaheuristic which uses some mechanisms inspired by the biological evolution: reproduction, mutation, crossover, and selection, see e.g., [14, 20]. Genetic algorithms have been already successfully applied for solving assignment problems for production lines, e.g., manual assembly U-lines [2], mixed-model assembly lines [12], two-sided assembly lines [15], robotic lines [17], disassembly lines [19].

The idea is to reproduce the evolution of populations where each individual is an encoded solution to the studied problem. Here, an algorithm using a non-binary based encoding like in [2] is suggested, i.e., the way to obtain a feasible solution is encoded and not a known solution. Actually, the RCL sequences are used as encoded individuals. In this way, the value of a gene i gives the number of the method to construct ith list RCL in GBL heuristic. The fitness value of an individual is the cost of the solution calculated with GBL heuristic using the corresponding RCL sequence.

An initial population of individuals (RCL sequences) is randomly generated. Two individuals are chosen from the population by the s-tournament method: twice s individuals are taken by random from the population and the one with the best fitness is selected. These two individuals become the parents for a new child.

The steady-state scheme of the GA is used. It means that the population size remains constant during the execution of the algorithm. One child is obtained from two parents. To obtain a child, *reproduction* operator is used consisting of *crossover* and *mutation* procedures.

In used crossover procedure, 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 value of each gene between them. The recombination mechanism is shown in Fig. 6. The general scheme is presented by Algorithm 1.

If the fitness value of the worst individual in the population is greater than the fitness value of a new child c, then the child replaces it. Each time the number of iterations during which the current best solution is not improved reaches the given limit, a new initial population is generated.

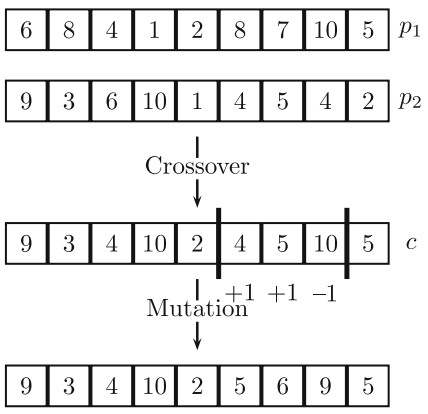

Fig. 6 Recombination mechanism

Algorithm 1: Steady-state scheme of the GA

```
repeat
    I_nimp ← 0
    Generate a new population
    repeat
        Select two parents p1 and p2 from the population
        by the s-tournament method
        Produce an offspring c by applying mutation and
        crossover to p1 and p2
        Choose the worst individual in the population
        and replace it by c
    until I_nimp > I_nimp_max
    Improvement phase
until T_tot > T_max
```

The same improvement phase as in Section 3.4 is applied to the best solution of each created population. This continues during the available execution time. The best found solution is returned as the final output. The main advantage of the presented approach lies in an easy recombination mechanism which does not require the verification of numerous constraints of the studied problem for each new individual.

In the next section, the two presented approaches are compared among them and other known methods for the studied problem.

5 Experimental results

In this section, the presented metaheuristic methods GRASP with PR (GBL-GR) and the genetic algorithm (GBL-GA) based on GBL heuristic are compared with the following heuristic and exact methods:

- the exact graph-based shortest path method [6], denoted by SP;
- CPLEX 11.1 MIP-solver applied to the problem with default solver settings, denoted by CPLEX [10].
- MIP-based GRASP (MIP-GR) and genetic algorithm (MIP-GA) presented in [3].

These methods are evaluated on two datasets. The first one is taken from [10]. It consists of five series of 50 benchmark problems. The second one contains two series with instances close to real industrial problems. These instances were generated taking into account real-life data of typical shapes of the parts manufactured in machining lines equipped with unit-built machines. Due to this, they represent relatively well real cases. The input data of the tests in GAMS format can be found at http://www.math.nsc.ru/AP/benchmarks/english.html.

The tests were carried out on Pentium-IV (3 GHz, 2.5 GB RAM). The algorithms are coded in C++. Both GBL-GA and MIP-GA were tested with the tournament size $s = 5$. The population size and the size of the elite set was equal to 20 for all problems.

In the presentation of the obtained results, the following notations are used: OS is the precedence constraints density, measured by the order strength of the transitive closure of graph G, NS is the number of instances for which feasible solutions were found; NO and NB are the number of instances where the optimal and the best-known solutions were obtained, respectively; $\Delta_{\max}$, Δ_{av} are respectively the percentage of maximal and average deviation of the cost of solutions from the optimal or the best-known ones;

Table 1 Results for series S3, $|\mathbf{N}| = 50$, OS = 90%, $m_0 = 10$

	SP	CPLEX	GBL-GR	MIP-GR	GBL-GA	MIP-GA
NS	50	50	50	50	50	50
NO	50	50	50	49	50	49
$\Delta_{\max}$	0	0	0	2	0	1.72
Δ_{av}	0	0	0	0.04	0	0.03
$T_{\max}$	0.09	463.4	300	300	300	300
T_{av}	0.04	41.1	300	300	300	300
$T_{\min}$	0.02	0.1	300	300	300	300

Table 2 Results for series S4, $|\mathbf{N}| = 50$, OS = 45%, $m_0 = 15$

	SP	CPLEX	GBL-GR	MIP-GR	GBL-GA	MIP-GA
NS	14	20	50	50	50	50
NB	14	13	44	46	41	48
$\Delta_{\max}$	–	–	3.23	13.15	6.45	2.7
Δ_{av}	–	–	0.35	0.46	0.59	0.11
$T_{\max}$	1,800	1,800	300	300	300	300
T_{av}	1,406	1,519	300	300	300	300
$T_{\min}$	12.9	31.5	300	300	300	300

$T_{\max}$, T_{av}, $T_{\min}$ are the maximal, average and minimal running times. T'_{av} is the average time till the final solution was found. Symbol "–" stands for unavailable data. The best result for a series is emphasized in bold.

5.1 Benchmark problem instances

The results for two first series S1–S2 from [10] are not presented here, since all tested methods found easily 100% of optimal solutions. For series S3–S5, the following input data are used: $C_1 = 10$, $C_2 = 2$, $\tau^{\text{b}} = \tau^{\text{m}} = 0$, $n_0 = 4$. For solving the medium size problems of series S3 and S4, the available time was limited by 1,800 s for the exact methods and by 300 s for the heuristics. The results for series S3 and S4 are reported in Tables 1 and 2.

For series S3, the shortest path method was the best one both in terms of the computation time and the quality of provided solutions. CPLEX also found the optimal solutions in all cases GBL-based metaheuristics reached the global optimum for each tested instance and are slightly better than MIP-based metaheuristics.

For series S4, the exact algorithms found feasible solutions in less than a half of the cases and there is no significant difference between them in terms of CPU times. Given the running time six times shorter, the heuristics were able to find feasible solutions in all cases. MIP-GA found the maximal number of best solutions, only two times another method provided a better solution. This result shows that metaheuristic methods are preferable for getting good solutions in

Table 3 Results for series S5, $|\mathbf{N}| = 100$, OS = 25%, $m_0 = 15$

	GBL-GR	MIP-GR	GBL-GA	MIP-GA
NB	10	37	5	36
$\Delta_{\max}$	33.33	30.77	33.33	12.82
Δ_{av}	4.63	1.63	5.01	1.46
T'_{av}	165.52	93.4	128.71	102.9

Table 4 Problem instances of series S6

Number	$\|N\|$	OS (%)	m_0	E^b (%)	E^m (%)	I^m (%)	GBL-GR	MIP-GR	GBL-GA	MIP-GA
1	68	43.6	34	41.3	32.2	0.83	26.5	28	26.5	28
2	71	51	35	41.4	30.1	0.56	30.5	32	30.5	32
3	78	38.7	39	46.6	34.2	0.47	46.5	47.5	46.5	47.5
4	71	53	35	45.4	27.9	1.65	25.5	27	25.5	27
5	72	42.1	36	36.2	39.8	0.31	34.5	35.5	34.5	34.5
6	74	49.8	37	46.9	26.1	0.81	36.5	38.5	37	38.5
7	71	49.7	35	40.2	31.9	0.85	27	28	27	27.5
8	75	47.1	37	38.6	32.8	0.65	25	26.5	25	26
9	75	49.5	37	39.6	36.4	0.22	40.5	44	41.5	42
10	76	41.1	38	44.2	32.4	1.12	35	38	35	37.5
11	81	42.8	40	48.4	28.3	0.19	45.5	47	45.5	45.5
12	92	46.5	46	42.7	28.7	0.74	44	45	43	45.5
13	71	52.7	35	45.5	31.6	0.56	38	38.5	37	38
14	65	42.8	32	50.1	27.6	0.63	37.5	39	37.5	39
15	46	40	23	44.9	29.2	0.39	29	29	29	29
16	74	42	37	40.3	33.8	0.22	42	44	42	43.5
17	74	45.8	37	38.5	42.3	0.15	42	45	42.5	45
18	70	38.6	35	41	37.6	0	43	44.5	43	44
19	69	44.5	34	40	32.7	1.15	30.5	32.5	30.5	31.5
20	64	38.9	32	38.2	38.8	0.89	22	22	21.5	22

short time in the case of a low constraints density for the medium size problems.

The results for series S5 are reported in Table 3. The available time was limited by 5,400 s (1.5 h) for the exact methods and by 600 s for the heuristics. Both of exact algorithms found less than ten solutions, this is why their results are excluded from the table. For this series, MIP-GR and MIP-GA demonstrated a rather similar behavior and it is difficult to choose the best one, while other metaheuristics are definitely inferior even if they found better solutions for several instances. It can be explained by the fact that the shortest path method that they use in the local search is relatively slow if the density of constraints is low [10]. Thus,

Table 5 Problem instances of series S7

Number	$\|N\|$	OS (%)	m_0	E^b (%)	E^m (%)	I^m (%)	GBL-GR	MIP-GR	GBL-GA	MIP-GA
1	99	41.3	49	45.6	32.5	0.56	40.5	41	40.5	40.5
2	111	45.7	55	43.5	36.5	0.16	66	68	66	67
3	94	44.3	47	40.1	38	0.43	48.5	50.5	48.5	50
4	122	43.8	61	39	34.5	0.24	58.5	60	58.5	58.5
5	105	47.8	52	35.1	42.5	0.18	49.5	52	49.5	50.5
6	97	45.9	48	45.7	34.6	0.32	49.5	51	50	50.5
7	101	36.9	50	41.9	31	0.44	43.5	45.5	43.5	44.5
8	87	32.9	43	38.1	36.5	0.32	45	47	45	46
9	110	40.3	55	39.7	34.6	0.8	35.5	35.5	35.5	35.5
10	113	45.9	56	44	36	0.22	65	66.5	65.5	66
11	113	39.5	56	39.4	37.5	0.49	36	38.5	35.5	38
12	96	49.5	48	39.8	31.4	0.64	34	37	34	36
13	116	47.3	58	35	40.6	0.39	49	50	49	50.5
14	121	44.7	60	37.2	32.2	0.34	46	49	46	47.5
15	111	39.7	55	39.3	32.9	0.74	37	40	37	38
16	98	35.8	49	46.7	33.7	0.19	45.5	46	45.5	45.5
17	114	48.2	57	37.4	37.6	0.56	49.5	50	49.5	50
18	99	44	49	38.3	34.1	0.45	38.5	40	38.5	39.5
19	119	44.5	59	37.2	38.5	0.58	42	43.5	41.5	43
20	127	40.7	63	50.4	28.1	0.4	64	64.5	64	64.5

Table 6 Results for series S6

	GBL-GR	MIP-GR	GBL-GA	MIP-GA
NB	17	1	17	3
Δ_{max}	2.7	8.64	2.47	7.14
Δ_{av}	0.37	4.71	0.25	3.6
T'_{av}	410.25	360.4	429.81	423.57

MIP-based metaheuristics can be clearly recommended for solving large-scale problems with a low density of constraints.

5.2 Industrial problem instances

Series S6–S7 contain 40 instances where operations are given by parameters λ_j, s_j and not by their times like in S3–S5. Here, $C_1 = 1, C_2 = 0.5, \tau^b = 0.2, \tau^m = 0.4, n_0 = 4$. Tables 4 and 5 present the generated instances and the solutions provided for them by the tested methods. Tables 6 and 7 summarize the obtained results. The exact methods were able to solve within 5,400 s only several instances from these datasets, so their performances are not displayed. The computational time given to the heuristics was 900 s for S6 and 3,000 s for S7.

The results in Tables 6 and 7 show that GBL-based metaheuristics are definitely more attractive for solving industrial case problems. For the two last datasets containing instances close to real industrial problems, GBL-GR and GBL-GA provided the best-known solution for all tested instances and outperformed all other tested methods. These approaches can be recommended for solving more complex instances where operations are given by machining parameters λ_j, s_j.

The obtained results demonstrate that all evaluated metaheuristics can provide good solutions (see Fig. 7). However, they should be applied in different cases. MIP-based metaheuristics (for example, MIP-GA) can be recommended for solving medium- and large-scale problems with a low level of constraints. On the other hand, the instances where operations are given by machining parameters must be solved by GBL-based metaheuristics (for example, GBL-GA) to get better solutions.

Table 7 Results for series S7

	GBL-GR	MIP-GR	GBL-GA	MIP-GA
NB	18	1	18	4
Δ_{max}	1.41	8.82	1.01	7.04
Δ_{av}	0.13	3.8	0.09	2.23
T'_{av}	1,213.4	1,603	1,155.47	1,323.5

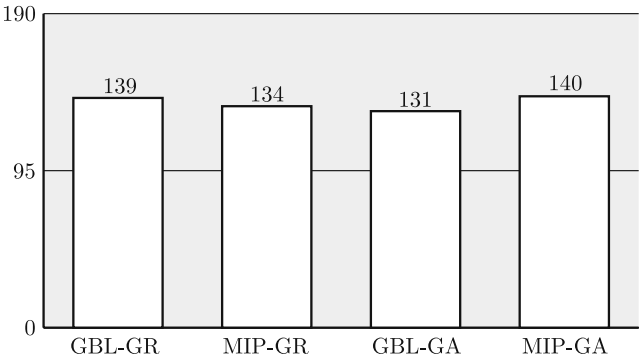

Fig. 7 Number of found best solutions per method

6 Conclusion

The problem studied in this article appears at the preliminary design stage of machining lines with unit-built machines. This problem consists in assigning a given set of operations to machines and spindle heads with the goal to reduce the number of equipment required for executing all operations under known technological and economical constraints.

This problem is NP-hard and powerful metaheuristic methods were needed for solving industrial size problems. In this paper, two new metaheuristic approaches have been presented and compared: one of them are based on GRASP and another one uses GA principles. Both methods (GBL-GR and GBL-GA) employ GBL heuristic for constructing initial feasible solutions and a local search phase with problem decomposition and solving subproblems by the shortest path method.

The performances of these metaheuristics have been compared with other existing methods on a dataset of academic benchmark problems and a dataset of industrial size instances. The obtained results indicate that the method to be used should be chosen in function of the parameters of a problem to solve. Large-scale problems with a low level of constraints should be solved by MIP-based metaheuristics (for example, MIP-GA). The use of GBL-based metaheuristics (for example, GBL-GA) gives more advantages in solving the instances where operations are given by machining parameters and/or the level of exclusion constraints among operations is rather high.

The future research should provide efficient algorithms for calculating lower bound for the considered problem. Actually, it is not a simple issue since the value of the objective function depends on the number of blocks opened at each machine, and this implies solving jointly a graph coloration and a line balancing prob-

lem. Another perspective is considering the stability of solutions provided by metaheuristics. Such an analysis may help designers to choose a more stable (under small changes in the input data) solution from a set of configurations provided by the used solution method.

Appendix A: Self-adjusting parameter α

Formulae (3), (4) use parameter α which controls the random part in the selection of an operation to be assigned. In the suggested algorithm, a mechanism of self-adjusting is applied. Instead of being chosen from a uniform distribution, the value of α is chosen from a discrete predefined set αSet. The probability of selecting a value of α is pr_α. When the algorithm starts, all pr_α are identical:

$$pr_\alpha = \frac{1}{|\alpha\text{Set}|}, \forall \alpha \in \alpha\text{Set}.$$

Every I_α iterations these probabilities are recalculated in order to favor the values of α that produce good feasible solutions. Let $C(S)$ be the cost of a feasible solution S. To measure the impact of a value of α on the quality of obtained solutions, the following data are used:

- the set of solutions obtained with a fixed value of α, denoted as S_α;
- the set of all known solutions, denoted as $\mathbf{S}$;
- the average solution cost obtained with a fixed value of α, denoted as $\mathtt{av}\{ C(S) \mid S \in S_\alpha\}$;
- the best-known solution cost, denoted as $C_{\min} = \min\{ C(S) \mid S \in \mathbf{S}\}$;
- the worst-known solution cost, denoted as $C_{\max} = \max\{ C(S) \mid S \in \mathbf{S}\}$.

At first, values val_α are calculated for all $\alpha \in \alpha$Set:

$$\text{val}_\alpha = \left(\frac{C_{\max} - \text{av}\{ C(S) \mid S \in S_\alpha\}}{C_{\max} - C_{\min}} \right)^\sigma,$$

where σ is a control parameter which is used to reduce the deviation between different values of α. Using the values of val_α, probabilities pr_α are calculated in the following way:

$$pr_\alpha = \frac{\text{val}_\alpha}{\sum_{\alpha \in \alpha\text{Set}} \text{val}_\alpha}.$$

Thus, new values of probabilities pr_α are increased for such values of α that produce good feasible solutions.

References

1. Andrés C, Miralles C, Pastor R (2008) Balancing and scheduling tasks in assembly lines with sequence-dependent setup times. Eur J Oper Res 187(3):1212–1223
2. Baykasoğlu A, Özbakır L (2007) Stochastic U-line balancing using genetic algorithms. Int J Adv Manuf Technol 32 (1–2):139–147
3. Dolgui A, Eremeev A, Guschinskaya O (2009) MIP-based GRASP and genetic algorithm for balancing transfer lines. In: Maniezzo V, Stützle T, Voß S. (eds) Matheuristics: hybridizing metaheuristics and mathematical programming. Annals of information systems, vol 10. Springer, New York. pp. 189–208
4. Dolgui A, Finel B, Guschinsky N, Levin G, Vernadat F (2006) MIP approach to balancing transfer lines with blocks of parallel operations. IIE Trans 38(10):869–882
5. Dolgui A, Guschinsky N, Levin G (2006) A special case of transfer lines balancing by graph approach. Eur J Oper Res 168(3):732–746
6. Dolgui A, Guschinsky N, Levin G, Proth JM (2008) Optimisation of multi-position machines and transfer lines. Eur J Oper Res 185(3):1375–1389
7. Feo T, Resende M (1989) A probabilistic heuristic for a computationally difficult set covering problem. Oper Res Lett 8(2):67–71
8. Festa P, Resende M (2002) GRASP: an annotated bibliography. In: Ribeiro C, Hansen P (eds) Essays and surveys on metaheuristics. Kluwer, Norwell. pp. 325–367
9. Glover F (1996) Tabu search and adaptive memory programming—advances, applications and challenges. In: Barr R, Helgason R, Kennington J (eds) Interfaces in computer science and operations research. Kluwer, Norwell. pp. 1–75
10. Guschinskaya O, Dolgui A (2009) Comparison of exact and heuristic methods for a transfer line balancing problem. Int J Prod Econ 120(2):276–286
11. Guschinskaya O, Dolgui A, Guschinsky N, Levin G (2008) A heuristic multi-start decomposition approach for optimal design of serial machining lines. Eur J Oper Res 189(3):902–913
12. Haq A, Rengarajan K, Jayaprakash J (2006) A hybrid genetic algorithm approach to mixed-model assembly line balancing. Int J Adv Manuf Technol 28(3–4):337–341
13. Hart J, Shogan A (1987) Semi-greedy heuristics: an empirical study. Oper Res Lett 6(3):107–114
14. Holland J (1975) Adaptation in natural and artificial systems. University of Michigan Press, Ann Arbor
15. Kim Y, Song W, Kim J (2009) A mathematical model and a genetic algorithm for two-sided assembly line balancing. Comput Oper Res 36(3):853–865
16. Laguna M, Martí R (1999) GRASP and path relinking for 2-layer straight line crossing minimization. INFORMS J Comput 11(1):44–52
17. Levitin G, Rubinovitz J, Shnits B (2006) A genetic algorithm for robotic assembly line balancing. Eur J Oper Res 168(3):811–825

18. Martino L, Pastor R (2010) Heuristic procedures for solving the general assembly line balancing problem with setups. Int J Prod Res 48(6):1787–1804
19. McGovern S, Gupta S (2007) A balancing method and genetic algorithm for disassembly line balancing. Eur J Oper Res 179(3):692–708
20. Reeves C (1997) Feature article—genetic algorithms for the operations researcher. INFORMS J Comput 9(3):231–250
21. Resende M, Pitsoulis L, Pardalos P (2000) Fortran subroutines for computing approximate solutions of MAX-SAT problems using GRASP. Discrete Appl Math 100(1–2):95–113
22. Resende M, Ribeiro C (2003) Greedy randomized adaptive search procedures. In: Glover F, Kochenberger G (eds) Handbook of metaheuristics. Kluwer, Norwell. pp. 219–249