Chapter 7
MIP-based GRASP and Genetic Algorithm for Balancing Transfer Lines

Alexandre Dolgui, Anton Eremeev, and Olga Guschinskaya

Abstract In this chapter, we consider a problem of balancing transfer lines with multi-spindle machines. The problem has a number of distinct features in comparison with the well-studied assembly line balancing problem, such as parameterized operation times, non-strict precedence constraints, and parallel operations execution. We propose a mixed-integer programming (MIP)-based greedy randomized adaptive search procedure (GRASP) and a genetic algorithm (GA) for this problem using a MIP formulation. Both algorithms are implemented in GAMS using the CPLEX MIP solver and compared to problem-specific heuristics on randomly generated instances of different types. The results of computational experiments indicate that on large-scale problem instances the proposed methods have an advantage over the methods from literature for finding high quality solutions. The MIP-based recombination operator that arranges the elements of parent solutions in the best possible way is shown to be useful in the GA.

7.1 Introduction

The problem considered in this chapter consists in balancing a transfer line where multi-spindle transfer machines without intermediate buffers are used. This problem is referred to as the *transfer line balancing problem* (TLBP) [5]. Machining transfer lines are usually paced and serial. They consist of a sequence of stations linked by an automated material handling device. In

Alexandre Dolgui · Olga Guschinskaya
Ecole Nationale Supérieure des Mines de Saint Etienne, Saint Etienne, France
e-mail: {dolgui,guschinskaya}@emse.fr

Anton Eremeev
Omsk Branch of Sobolev Institute of Mathematics SB RAS, Omsk, Russia
e-mail: eremeev@ofim.oscsbras.ru

V. Maniezzo et al., (eds.), *Matheuristics*, Annals of Information Systems 10,
DOI 10.1007/978-1-4419-1306-7_7,

lines of this type, each station is equipped by a special machine-tool which performs machining operations block by block. All operations of each block are executed simultaneously using one multi-spindle head. The parallel execution of operations in a block is possible due to the fact that the multi-spindle heads carry several simultaneously activated tools. When machining at the current station is finished (all blocks installed on this machine have been activated) the part is moved to the next station. The time span between two movements can not exceed the given time value T_0 referred to as *line cycle time*. The balancing problem consists in assigning the given set of operations to parallel blocks and stations under given assignment restrictions.

The line balancing problem in the assembly environment is well-studied in the literature. Several reviews of different formulations and used solution methods are available e.g. in [1, 2, 9]. The TLBP has a number of unique characteristics such as parameterized operation times, non-strict precedence constraints, and parallel operations execution. These features make it impossible to use directly the optimization methods developed for assembly line balancing problems; for details see [5]. Several exact (e.g. mixed-integer programming and graph approaches) and heuristic (e.g. FSIC (First Satisfy Inclusion Constraints) and multi-start decomposition algorithm) methods have been developed for the TLBP. A description of these methods is given in [10]. Later, in [11] it was proposed to use greedy randomized adaptive search procedures (GRASP) for solving this problem.

In this chapter, we propose a MIP-based greedy randomized adaptive search procedure (GRASP) and a genetic algorithm (GA) for the TLBP, using the MIP formulation [5] of the TLBP in both algorithms. The solution construction and the local improvement stages of GRASP are based on solving sub-problems of smaller size. The same solution construction method is used for building the initial population in the GA. The crossover and mutation in the GA are combined in a MIP-recombination operator, similar to the recombination proposed in [3].

Both algorithms are implemented in GAMS using the CPLEX MIP solver and compared to problem-specific heuristics [10] on randomly generated instances of different types. The results of computational experiments indicate that on large problem instances the methods proposed offer an advantage over the methods from literature in finding high quality solutions. The capability of the MIP-recombination operator to arrange the elements of parent solutions in the best possible way is shown to be useful in the GA.

The chapter is organized as follows. The problem statement in the MIP formulation is given in Section 7.2. The solution methods are discussed in Sections 7.3 and 7.4. The results of computational experiments are presented in Section 7.5. Concluding remarks are given in Section 7.6.

7.2 Problem Statement

For the TLBP the following input data are assumed to be given [5]:

- $\mathbf{N}$ is the set of all operations involved in machining of a part;
- T_0 is the maximal admissible line cycle time;
- τ^S and τ^b are the auxiliary times needed for activation of a station and a spindle head (block), respectively;
- C_1 and C_2 are the relative costs of one station and one spindle head (block);
- m_0 is the maximal admissible number of stations;
- n_0 is the maximal number of spindle heads (blocks) per station;
- *Precedence constraints* between the operations. These constraints define a non-strict partial order relation over the 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 the set D if and only if the block with operation j cannot precede the block with operation i. (If $(i, j) \in D$ then the 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 station, because of a required machining tolerance. These constraints can be represented by a family ES of subsets of $\mathbf{N}$, such that all operations of the same subset $e \in ES$ must be assigned to the same station;
- *Station exclusion constraints* defining the groups of operations that cannot be assigned to the same station because of their technological incompatibility. These constraints are represented by a family $\overline{ES}$ of subsets of $\mathbf{N}$, such that all elements of the same subset $e \in \overline{ES}$ cannot be assigned to the same station.
- *Block exclusion constraints* defining the groups of operations that cannot be assigned to the same block because of their technological incompatibility. These constraints are represented by a family $\overline{EB}$ of subsets from $\mathbf{N}$, such that all elements of the same subset $e \in \overline{EB}$ cannot be assigned to the same block.
- For each operation j, its processing time t_j is given 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.

A MIP formulation for solving the TLBP was suggested in [5]. Here, we reproduce this model with the improvements proposed in [10].

Let us denote the set of all operations assigned to station k by N_k and let the set of operations grouped into block l of station k be N_{kl}. Station processing time $t^S(N_k)$ equals the sum of its block processing times: $t^S(N_k) = \sum_{l=1}^{n_k} t^b(N_{kl}) + \tau^S$.

We will consider two definitions of block processing time. A simplified definition [10] uses the assumption that the block processing time $t^b(N_{kl})$ is

equal to the duration of the longest operation in the block:

$$t^b(N_{kl}) = \max\{t_j | j \in N_{kl}\} + \tau^b. \tag{7.1}$$

A more general definition [12] does not use the processing times of operations t_j, but rather the parameters λ_j and s_j:

$$t^b(N_{kl}) = \frac{\max\{\lambda_i | i \in N_{kl}\}}{\min\{s_i | i \in N_{kl}\}} + \tau^b. \tag{7.2}$$

Note that the latter definition covers the first case, assuming $\lambda_j = t_j,\ s_j = 1$ for all j. Besides that, it suits better the practical situations where the operations assigned to several tools fixed within one block may require different depths of their cuts and different maximal admissible feed speeds.

In the MIP formulation below we use the following notation:

- q is the block index; $q = (k-1)n_0 + l$ is the l-th block of a station k;
- q_0 is the maximal possible value of q, $q_0 = m_0 n_0$;
- $S(k) = \{(k-1)n_0 + 1, \ldots, kn_0\}$ is the set of block indices for a station k;
- $Q(j)$ is the set of indices q (blocks) where operation j can be assigned;
- $K(j)$ is the set of indices k (stations) where operation j can be assigned;
- e is a set of operations which is an element of ES, $\overline{ES}$ or $\overline{EB}$;
- $j(e)$ is an arbitrarily fixed operation from the set e.
- t_j is the execution time of operation j if it is performed alone in a block. In the simplified formulation, t_j is given as input data. Otherwise, $t_j = \frac{\lambda_j}{s_j}$.
- $t_{ij} = \frac{\max\{\lambda_i, \lambda_j\}}{\min\{s_i, s_j\}}$ is the execution time of two operations i, j if they are performed in one block. In the simplified formulation this value is not used.

The following variables will be involved:

- X_{jq} is a binary decision variable (1 if operation j is assigned to block q and 0 otherwise);
- F_q is an auxiliary real-valued variable for determining the time of processing the block q;
- Y_q is an auxiliary binary variable that indicates if the block q exists;
- Z_k is an auxiliary binary variable that indicates if the station k exists.

The variables Y_q and Z_k are used to count the number of blocks and stations, respectively. To reduce the number of decision variables and constraints, the sets of possible block and station indices $Q(j)$, $K(j)$ for each operation j are obtained by means of the procedure described in [5].

The problem consists in the minimization of investment costs incurred by construction of stations and spindle heads (blocks):

$$\text{Min } C_1 \sum_{k=1}^{m_0} Z_k + C_2 \sum_{q=1}^{q_0} Y_q \tag{7.3}$$

subject to

$$\sum_{q \in Q(j)} (q-1) X_{jq} \geq \sum_{s \in Q(i)} (s-1) X_{is}, \quad (i,j) \in D, \tag{7.4}$$

$$\sum_{q \in Q(j)} X_{jq} = 1, \quad j \in \mathbf{N}, \tag{7.5}$$

$$\sum_{j \in e \setminus \{j(e)\}} \sum_{q \in S(k) \cap Q(j)} X_{jq} = (|e|-1) \sum_{q \in S(k)} X_{j(e)q}, \quad e \in ES,\ k \in K(j(e)), \tag{7.6}$$

$$\sum_{j \in e} X_{jq} \leq |e| - 1, \quad e \in \overline{EB},\ q \in \cap_{j \in e} Q(j), \tag{7.7}$$

$$\sum_{j \in e} \sum_{q \in S(k) \cap Q(j)} X_{jq} \leq |e| - 1, \quad e \in \overline{ES},\ k \in \cap_{j \in e} K(j), \tag{7.8}$$

$$F_q \geq (t_i + \tau^b) X_{iq}, \quad i \in \mathbf{N},\ q \in Q(i), \tag{7.9}$$

$$F_q \geq (t_{ij} + \tau^b)(X_{iq} + X_{jq} - 1), \quad i,j \in \mathbf{N},\ i < j,\ q \in Q(i) \cap Q(j), \tag{7.10}$$

$$\tau^S + \sum_{q \in S(k)} F_q \leq T_0, \quad k = 1, 2, \ldots, m_0, \tag{7.11}$$

$$Y_q \geq X_{jq}, \quad j \in \mathbf{N},\ q \in Q(j), \tag{7.12}$$

$$Z_k = Y_{(k-1)n_0+1}, \quad k = 1, 2, \ldots, m_0, \tag{7.13}$$

$$Y_{q-1} - Y_q \geq 0, \quad q \in S(k) \setminus \{(k-1)n_0 + 1\},\ k = 1, 2, \ldots, m_0, \tag{7.14}$$

$$Z_{k-1} - Z_k \geq 0, \quad k = 2, 3, \ldots, m_0, \tag{7.15}$$

$$X_{jq},\ Y_q,\ Z_k, \in \{0,1\}, \quad j \in \mathbf{N},\ q = 1, 2, \ldots, q_0,\ k = 1, \ldots, m_0, \tag{7.16}$$

$$F_q \in [0, T_0 - \tau^S - \tau^b], \quad q = 1, 2, \ldots, q_0. \tag{7.17}$$

Here inequalities (7.4) impose the precedence constraints; equalities (7.5) reflect the fact that each operation must be assigned to exactly one block; constraints (7.6) determine the necessity of grouping certain operations in the same station; constraints (7.7)-(7.8) deal with the impossibility of grouping certain operations in one block or executing certain operations at the same station, respectively; constraints (7.9) and (7.10) determine the block processing times according to (7.1) or (7.2): here condition (7.9) corresponds to the case of a single operation in a block, while (7.10) covers the cases of two or more operations (note that in the simplified formulation with block time defined by (7.1), inequality (7.10) is redundant); constraint (7.11) imposes the bound on cycle time; constraints (7.12) ensure that block q exists in the design decision if and only if $X_{jq} = 1$ for some j; equalities (7.13) ensure that a station k exists in the design decision if and only if at least one block is assigned to it; constraints (7.14) guarantee that block q is created in station k only if block $q-1$ exists for this station; constraints (7.15) ensure that station k can be created only if station $k-1$ is created.

Inequalities (7.14) and (7.15) mainly serve as symmetry-breaking cuts in this model (note that by a simple modification of (7.13) one could make these inequalities redundant). Bounds (7.16) are also imposed to reduce the polyhedron of the linear relaxation. One could assume that all variables Y_q and Z_k are real values from the interval [0,1] but we do not use this assumption, because our preliminary experiments indicate that binary variables Y_q and Z_k yield a more appropriate problem formulation for the CPLEX MIP solver.

Another modification that can improve the performance of branch-and-cut algorithms consists in adding a relatively small penalty term to the objective function. The greater is the block number where an operation is assigned, the greater penalty is given:

$$\text{Min } C_1 \sum_{k=1}^{m_0} Z_k + C_2 \sum_{q=1}^{q_0} Y_q + C_3 \sum_{q=1}^{q_0} \sum_{j \in \mathbf{N}} (1-q) X_{jq}. \tag{7.18}$$

Here, the weight C_3 may be chosen sufficiently small, so that the optimal solution of the modified problem (7.4)–(7.18) is also optimal for problem (7.3)–(7.17). The penalty term breaks some symmetries of the problem and provides appropriate bias when the branching is made. Our experiments indicate that for faster search of approximate solutions the value C_3 may be chosen adaptively (see the details in Section 7.5).

7.3 Greedy Randomized Adaptive Search Procedure

GRASP is a multi-start metaheuristic algorithm, where each iteration consists of two phases: constructing a feasible solution and improving it. Both phases are repeated interchangeably until a stopping criterion is satisfied. Extensive bibliographies on GRASP were published in [8, 16]. The general scheme of GRASP is as follows:

GRASP method
Until a stopping criterion is satisfied do:
1. Construct a random feasible solution.
2. If a feasible solution is constructed,
   apply a local improvement procedure to it.
3. Update the best found solution.

7.3.1 Construction Phase

In the case of the TLBP, a feasible solution at Step 1 can be obtained by applying a randomized greedy heuristic algorithm (see, e.g., [13]). The MIP-based greedy algorithm for the TLBP starts from a transfer line with an empty set of blocks, and then the blocks are created consecutively. A feasible solution is constructed by adding a set N^t_{add} of one or more operations at each step t of the greedy algorithm. We will denote by N^t the set of operations that have been assigned to stations on steps $1, \ldots, t$ assuming $N^0 = \emptyset$.

The set N^t_{add} is constructed by a randomized procedure which has two tunable parameters $\alpha \in [0, 1]$ and $\beta \in \{1, \ldots, |\mathbf{N}|\}$. Adjustment of α and β will be discussed in Section 7.5. At the beginning of step t it is assumed that $N^t_{add} = \emptyset$, then it is extended in the following loop:

1. Compute the *set of candidate operations* N_{CL}, consisting of all operations that can be allocated after the set of operations $N^{t-1} \cup N^t_{add}$ in view of inclusion and precedence constraints. To this end, we use a supplementary digraph $G' = (\mathbf{N}, D')$ which is obtained from G by adding the arcs (i, i') for all pairs i, i' such that $i \in e,\ i' \in e$ for some $e \in ES$, and by taking the transitive closure of this extended digraph. The inclusion and precedence constraints will not be violated if a new operation j is executed after all operations of the set $N^{t-1} \cup N^t_{add}$, provided that $N^{t-1} \cup N^t_{add}$ contains all such i that $(i, j) \in D'$ and $(j, i) \notin D'$. The graph G' may be computed by means of Warshall's algorithm in time $O(|\mathbf{N}|^3)$, before the GRASP iterations begin.
2. Rank the operations in N_{CL} according to the values of *greedy function* $g(j)$ (this function will be described later). Find

$$g_{max} = \max\{g(j) : j \in N_{CL}\} \text{ and } g_{min} = \min\{g(j) : j \in N_{CL}\}.$$

3. Place the well-ranked candidate operations j with $g(j) \geq g_{max} - \alpha(g_{max} - g_{min})$ into a set N_{RCL}, called *restricted candidate list*. The parameter α controls the trade-off between randomness and greediness in the construction process.
4. Select an element j uniformly at random from N_{RCL} and add j to the set N^t_{add}.
5. Include into N^t_{add} all operations i such that $(i, j) \in D'$ and $(j, i) \in D'$ (the precedence and inclusion constraints imply that these operations must be placed in the same block with j).

This loop continues until β iterations are made or there are no more operations to add, i.e. $N^{t-1} \cup N^t_{add} = \mathbf{N}$.

The iterations of the greedy algorithm continue until either all operations are assigned and a feasible solution is obtained or it is impossible to create a new station since $m + 1 > m_0$.

The greedy function $g(j)$ measures the impact of assigning operation j at the current iteration. Several greedy heuristics were elaborated for the simple assembly line balancing problem with the greedy functions based on priority rules; see, e.g., [17]. However, all previously considered priority rules are based on the hypothesis that all operations are executed sequentially and the operation times are cumulated. This hypothesis is not right for the TLBP, where operations can be executed in parallel. As a consequence, the greedy function can hardly be based on the known priority rules. We use a simple greedy function $g(j)$ equal to the lower bound on the number of blocks required to assign all successors of operation j. This lower bound is calculated by the algorithm suggested in [5].

Once the set N^t_{add} is chosen, the operations of this set are appended to the current partial solution which has been computed on the previous iterations. Let us define for all $j \in \mathbf{N}$, $q = 1, \ldots, q_0$ the set of values $x^{(t)}_{jq}$, such that $x^{(t)}_{jq} = 1$ if operation j is assigned to block q in the partial solution obtained at iteration t, and $x^{(t)}_{jq} = 0$ otherwise. Allocation of the new operations can be carried out by means of a supplementary MIP problem. This MIP problem is formulated by the set of constraints (7.4)–(7.17) and the objective function (7.18) but a large number of binary variables are fixed equal to zero as described below.

Let k_{use} denote the number of the last station to which operations have been assigned at the latest partial solution, i.e.

$$k_{use} = \max\left\{ k \mid \sum_{j \in \mathbf{N}} \sum_{q \in S(k)} x^{(t-1)}_{jq} \geq 1 \right\}.$$

We aim to allocate the operations of the set N^t_{add} to the stations with numbers not greater than $k_{max} = k_{use} + \beta$ (this is always possible if the problem is solvable and $k_{max} \leq m_0$). To this end, we do not fix the variables X_{jq} with $j \in N^t_{add}$ and $q \leq q_{max}$, where $q_{max} = k_{max} n_0$. We also do not fix the variables X_{jq} such that $x^{(t-1)}_{jq} = 1$ or $q = 1 + \max\{q | \sum_{j \in \mathbf{N}} x^{(t-1)}_{jq} \geq 1\}$ to allow some previously allocated operations to be moved into the first new block, if it allows to save the cost. All the rest of the variables X_{jq} are fixed to zero value. The resulting sub-problem at each step t of the greedy heuristic is solved by a MIP solver. The value of parameter β is chosen experimentally so that the resulting sub-problems involve as many operations as possible, but the computational cost of the MIP solver in each iteration t is "not too large".

7.3.2 Improvement Phase

The improvement heuristic starts with a feasible solution obtained at the construction phase in order to improve it. For this purpose, a MIP-based modification of the *decomposition algorithm with aggregate solving of sub-problems* (DAASS) [12] is used. The algorithm DAASS has already been used with heuristic FSIC [4], which constructs a feasible solution without applying any greedy function, in [10, 12].

The decomposition consists in cutting the sequence of stations corresponding to the given feasible solution into several non-intersecting subsequences. The size of each subsequence is chosen at random, as it is described below. The total number w of such subsequences is known only at the end of the decomposition procedure. A subsequence $K_r,\ r = 1, \ldots, w$ involving a random number of stations k_r, is used to generate a sub-problem SP_r.

In DAASS, each sub-problem SP_r is solved exactly by the graph approach described in [6]. The results of previously solved sub-problems in DAASS are taken into account while solving the next sub-problem: each block of operations existing in the solution to SP_r is replaced by a *macro-operation* and all macro-operations are included in the consecutive sub-problem SP_{r+1}.

In contrast to the DAASS method, the local improvement procedure used in the present chapter is based on solving a series of sub-problems in MIP formulation (7.4)–(7.18). To simplify the algorithm, at step $r,\ r = 1, \ldots, w$ we do not construct the macro-operations in this heuristic, but simply fix all binary variables non-related to the stations of set K_r, equal to the corresponding values of the best found solution. The remaining variables $X_{jq},\ Y_q,\ q \in \cup_{k \in K_r} S(k)$, and $Z_k, k \in K_r$ are optimized by a MIP solver.

In the randomized choice of the sets K_r we have to ensure that on one hand, the size of a sub-problem is not "too small" and the solver can often improve the heuristic solution, on the other hand, the size of a sub-problem is not "too large" and it is possible to apply the solver in reasonable CPU time.

The following parameters, are used to limit the size of the sub-problems [12]: (i) the maximal number of stations k_{max} within one subsequence, that is the maximum possible value of k_r; (ii) the maximal number of operations n_{max} within one subsequence.

The value k_r is chosen uniformly at random within $[1, k_{max}]$ and then can be modified so that the total number of operations in the sub-problem does not exceed N_{max} and $\sum_{r=1}^{w} k_r$ is not greater than the number of stations in the current heuristic solution.

7.4 Genetic Algorithm

A *genetic algorithm* is a random search method that models a process of evolving a population of individuals [14, 15]. Each individual corresponds to some solution of the problem (feasible or infeasible) and it is characterized by the *fitness* which reflects the objective function value and the satisfaction of problem constraints. The better the fitness value, the more chances are given for the individual to be selected as a parent. New individuals are built by means of a *reproduction* operator that usually consists of *crossover* and *mutation* procedures. The crossover procedure produces the offspring from two parent individuals by combining and exchanging their elements. The mutation procedure adds small random changes to an individual. The formal scheme of the GA with steady state replacement is as follows:

Steady-state scheme of the GA
1. Generate the initial population.
2. Assign $t := 1$.
3. Until a termination condition becomes true do:
   3.1 Selection: choose p_1, p_2 from the population.
   3.2 Produce a child c applying mutation and crossover to p_1 and p_2. The crossover is used with probability P_c.
   3.3 Choose the worst individual in population w.r.t. the fitness function and replace it by c.
   3.4 t:=t+1.
4. Result is the best found solution w.r.t. fitness function.

In our implementation of the GA the fitness function is identical with the objective function. The choice of each parent on Step 3.1 is done by the s-tournament selection: take s individuals at random from the population (uniformly distributed, repetitions allowed) and select the best one w.r.t. the objective function. The population size remains constant during the execution of the GA - this parameter is denoted by N_{ind}.

In each iteration of a steady-state GA, most of the individuals of the current population are kept unchanged, which is different from the canonical GA proposed by Holland [14]. In many implementations of the steady-state GA the offspring replace the worst individuals of the population, but there are alternative replacement rules as well [15].

In this chapter, we will assume that the GA is restarted every time the termination condition halts it. This continues until the overall execution time will reach the limit T. The best solution found over all runs is returned as the final output. In our computational experiments we have tested an alternative approach, where the GA runs for the whole period T without restarts (see Subsection 7.5.2) but it turned to be inferior to the GA with this restart rule.

Let θ be the iteration when the latest solution improvement took place. The termination condition of GA is: restart if during the last θ iterations there was no best-found solution improvement and $t > N_{ind}$.

Encodings of solutions in a GA are usually called *genotypes*. One of the most essential issues in the development of a GA is the choice of representation of solutions in genotypes. In the GA proposed in this chapter, the genotype consists of values of the binary variables X_{jq} which describe the whole assignment of operations. The genotypes of the initial population are generated at random by N_{ind} runs of the GRASP heuristic described in Section 7.3.

MIP-Recombination

As proposed in [3], we combine mutation and crossover into a *MIP-recombination* operator applied instead of Step 3.2 in the steady-state GA. In the case of TLBP the MIP-recombination operator consists in solving a MIP problem, which is obtained from the original problem (7.3)–(7.17) as follows:

MIP-recombination operator
1. Fix all Boolean variables equal to their values in p_1.
2. Release all Boolean variables where p_1 differs from p_2 (analog of crossover).
3. Release a random subset of fixed variables independently with probability P_m (analog of mutation).

In our implementation, the MIP-solver of CPLEX 11.1 is used to find the optimum of the sub-problem emerging in the MIP-recombination. To avoid time-consuming computations we set a time limit T_{rec} for each call to the solver. Unlike the standard GA crossover, the described MIP-recombination procedure produces only one new individual at each iteration. If the parent solutions are feasible, the solver always returns a feasible solution to the MIP-

recombination sub-problem because the genotype of one of the parents is sent to CPLEX as a MIP-start feasible solution.

The initial value of mutation parameter P_m is set to P_m^0 and adapted in the process of GA execution. Every time the MIP-recombination sub-problem is solved to optimality, the parameter P_m is multiplied by 1.1. Whenever the solver is unable to find an optimal solution within the time limit T_{rec}, and $P_m > P_m^0$, parameter P_m is divided by 1.1. This adaptive mechanism keeps the complexity of the MIP-recombination problems close to the limit where the solver is able to obtain exact solutions within the given time T_{rec}.

7.5 Experimental Results

In this section, we compare the genetic algorithm and MIP-based GRASP proposed in this chapter (further referred to as GA and MIP-GRASP, respectively) to the following three heuristic and exact methods:

- the multi-start hybrid decomposition algorithm (henceforth denoted by HD) combining DAASS [10, 12] with the FSIC heuristic, where FSIC is used for construction of a feasible solution;
- the exact graph-based shortest path method [6, 10], denoted by SP;
- CPLEX 11.1 MIP-solver applied to problem (7.3)–(7.17) with the default solver settings, denoted below by CPLEX.

7.5.1 Problem Instances

In the experiments, we use five test series S1-S5 from [10], each one containing 50 randomly generated instances and two new series S6 and S7, also randomly generated but with more realistic data sets. Both series S6 and S7 consist of 20 instances. The number of operations and the upper bound m_0 on the number of stations are shown in Table 7.1. Also, this table gives the precedence constraints density, measured by the order strength (OS) of graph G. Here, by order strength we mean the number of edges in the transitive closure of graph G divided by $|\mathbf{N}|(|\mathbf{N}|-1)/2$. In series S1-S5 we have $C_1 = 10, C_2 = 2$, $\tau^b = \tau^S = 0$, $n_0 = 4$. In S6 and S7, $C_1 = 1, C_2 = 0.5$, $\tau^b = 0.2, \tau^S = 0.4$, $n_0 = 4$.

The details on the random generation of series S1-S5 can be found in [10]. The series S6 and S7 consist of more realistic instances for two reasons. Firstly, they contain non-trivial input data for parameters λ_j, s_j, while series S1-S5 in effect consist of problems in simplified formulation defined in terms of operation times t_j. Secondly, in S6 and S7 the pseudo-random choice of operations was not performed independently and uniformly as in Series S1-S5, but based on real-life data of typical shapes of the parts manufactured in

mechanical transfer lines. The random choice is applied to the shape of parts, which further defines the parameters and mutual compatibility of operations. The input data of the benchmarks in GAMS format can be found in the Discrete Location Problems Benchmarks Library by `http://www.math.nsc.ru/AP/benchmarks/english.html`.

7.5.2 Experimental Settings

The experiments were carried out on a Pentium-IV computer (3 GHz, 2.5 Gb RAM). Both the GA and the MIP-GRASP were programmed in GAMS 22.8, the rest of the algorithms being considered were coded in C++.

The tunable parameters of the constructive heuristic in MIP-GRASP and in the GA initialization were chosen as follows: $\alpha = 0.25$, $\beta = 10$. This tuning was based on preliminary experiments with different values of α and β on series S5. The frequency of finding best-known objective as a function for different values of α and β in these trials can be seen on Figure 7.1. The 95% confidence intervals for probability of finding the best-known objective value are displayed for $\beta = 20$. It follows from this figure that $\beta = 20$ is preferable for series S5. We chose $\beta = 10$ for all subsequent experiments only because on large problems of series S6 and S7 usage of $\beta = 20$ leads to so hard sub-problems, that MIP-GRASP is sometimes unable to find a feasible solution in the given amount of time.

On the basis of similar considerations, the parameters of the improvement heuristic in MIP-GRASP were set to $k_{max} = 15$, $n_{max} = 50$.

We also found that the value of the penalty term C_3 has a statistically significant impact ($p < 0.05$) on the quality of MIP-GRASP results. In order to choose C_3 adaptively while solving a given instance in GA or MIP-GRASP, we set it initially to such a small value that it does not change the optimal line design (by solving two supplementary MIP problems). Further, parameter C_3 is optimized by a simple one-dimensional search routine. This is done in MIP-GRASP restarts or in the construction of the initial population for the GA, using the average quality of solutions of the constructive heuristic as the optimization criterion.

The CPLEX tolerance parameter `optca` in the greedy heuristic was set to 2; parameter `mipstart` was set to 1 in the improvement heuristic and in

Table 7.1 Testing series

Series	1	2	3	4	5	6	7
$\|\mathbf{N}\|$	25	25	50	50	100	46 - 92	94 - 125
OS	0.5	0.15	0.9	0.45	0.25	-	-
m_0	15	4	10	15	15	23 - 46	43 - 62

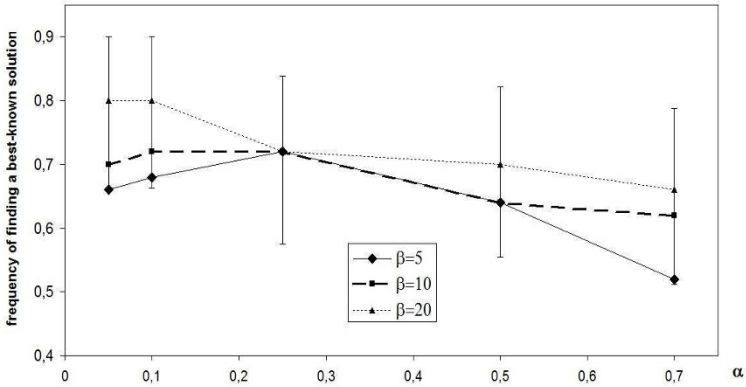

Fig. 7.1 Frequency of finding solutions with best-known objective value as a function of parameter α for different values of β in MIP-GRASP on series S5.

the MIP recombination operator, the rest of CPLEX options were set as the default. The termination conditions of the main loop in MIP-GRASP and in the GA were triggered by reaching the given CPU time limit (see the tables below).

In the experiments with the GA, we set the tournament size $s = 5$, and the initial mutation probability $P_m^0 = 0.1$. The time given for each call to the MIP-recombination operator was $T_{rec} = 5$ sec. The population size N_{ind} was set to 20 for all problems.

The GA restarting rule, described in Section 7.4, has been compared to straightforward running GA for the whole period T without restarts. The comparisons were carried out on the sets S3, S5 and S6, using the algorithmic settings described in Subsection 7.5.3 below. The GA with restarts obtained the best-known solutions more frequently on all three series. Besides that, on series S5, its advantage was shown to be statistically significant with level $p < 0.02$ in Wilcoxon matched pairs test. This motivated our choice of the restarting rule.

7.5.3 Results

In the presentation of the obtained results, we use the following notation: 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; Δ_{max}, Δ_{avg}, Δ_{min} are the percentage of maximal, average and minimal deviation of the cost of solution from the optimal or the best-known objective value, respectively; T_{max}, T_{av}, T_{min} are

Table 7.2 Results for series S1 and S2

	Series 1				Series 2			
	SP	CPLEX	HD	MIP-GRASP	SP	CPLEX	HD	MIP-GRASP
NS	50	50	50	50	50	50	50	50
NO	**50**	**50**	39	49	**50**	**50**	35	**50**
Δ_{max}	**0**	**0**	5.26	4.16	**0**	**0**	11.1	**0**
Δ_{av}	**0**	**0**	1.0	0.08	**0**	**0**	1.7	**0**
Δ_{min}	**0**	**0**	0.0	0.0	**0**	**0**	**0**	**0**
T_{max}	**11**	841	90	90	1638	**3.53**	90	90
T_{av}	**1.4**	38	90	90	292	**0.86**	90	90
T_{min}	**0.03**	0.39	90	90	3.77	0.06	90	90

the maximal, average and minimal running time. T'_{av} is the average time till the final solution was found. Symbol "-" stands for unavailable data. The best result of a series is emphasized in bold.

7.5.3.1 Small Sized Instances

For the first two series, the available computational time was limited to 1800 seconds for the exact methods and to 90 seconds for the heuristics. The results for series S1 and S2 are reported in Table 7.2. In both series the two exact algorithms found the optima for all instances. One can see that the shortest path method performs best on series S1 in terms of computational time.

For series S2, CPLEX is the best in terms of computational time. The MIP-GRASP found all optima, outperforming HD, but these heuristics were given more CPU time than CPLEX used. In fact the average time of finding the optimal solution for MIP-GRASP was also greater than the CPLEX time. The precision of MIP-GRASP is higher on this series than on series S1, which is due to better performance of CPLEX solver on problems with low order strength (see e.g. [10]). The GA was not tested on series S1 and S2 since they are simple even for MIP-GRASP.

7.5.3.2 Medium Sized Instances

For solving the medium sized instances of series S3 and S4, the available computational time was limited to 1800 seconds for the exact methods and to 300 seconds for the heuristics. The results for series S3 and S4 are reported in Tables 7.3 and 7.4.

On series S3, the shortest path method is again the best one both in terms of the computation time and the quality of provided solutions. CPLEX found the optimal solutions in all cases and MIP-GRASP missed the optimum twice. The average and maximal time of finding the final solution for MIP-GRASP

Table 7.3 Results for series S3

	SP	CPLEX	HD	MIP-GRASP	GA
NS	50	50	50	50	50
NO	50	50	28	48	49
Δ_{max}	**0**	**0**	4.2	2.27	1.72
Δ_{av}	**0**	**0**	1.0	0.09	0.03
Δ_{min}	**0**	**0**	0	0	0
T_{max}	**0.09**	463.4	300	300	300
T_{av}	**0.04**	41.1	300	300	300
T_{min}	**0.01**	0.1	300	300	300

Table 7.4 Results for series S4

	SP	CPLEX	HD	MIP-GRASP	GA
NS	14	20	**50**	**50**	**50**
NO	14	13	39	42	**48**
Δ_{max}	-	-	13.15	13.15	**2.7**
Δ_{av}	-	-	0.86	0.64	**0.11**
Δ_{min}	-	-	**0**	**0**	**0**
T_{max}	1800	1800	300	300	300
T_{av}	1490.3	1519.0	300	300	300
T_{min}	13.0	31.5	300	300	300

was 5.02 seconds and 66 seconds, respectively, which is much shorter than the given running time. The GA demonstrated a similar behavior, slightly outperforming MIP-GRASP. The results of the hybrid method HD are inferior to those of all other algorithms in terms of the solution quality.

In series S4, the exact algorithms found feasible solutions in less than half of the cases; see Table 7.4. The CPU times were similar for these methods. In contrast, the heuristics were able to find feasible solutions in all cases, given a six times shorter computation time. The overall quality of solutions is better in the case of the GA. In general, this table shows that in cases of low density of constraints even for the medium size problems the heuristic methods are preferable, when the computational time is limited.

7.5.3.3 Large Sized Instances

The results for series S5 are reported in Table 7.5. The available computational time was limited to 5400 seconds for the exact methods and to 600 seconds for the heuristics. Both exact algorithms found less than 10 solutions; therefore, they are excluded from the table. The heuristics found feasible solutions in all 50 cases. In this series MIP-GRASP and GA demonstrate similar behavior and it is hard to tell which one is better, while HD is definitely inferior.

Table 7.5 Results for series S5

	HD	MIP-GRASP	GA
NB	11	**37**	36
Δ_{max}	35.9	30.8	**12.8**
Δ_{av}	5.02	1.6	**1.5**
T'_{av}	-	**93.4**	102.9

Table 7.6 Results for series S6 and S7

	Series 6			Series 7		
	HD	MIP-GRASP	GA	HD	MIP-GRASP	GA
NB	10	8	**15**	6	2	**17**
Δ_{max}	7.6	5.55	**4.35**	6	5.26	**1.28**
Δ_{av}	1.76	1.59	**0.72**	2.26	1.87	**0.16**
T'_{av}	-	360.4	423.57	-	1603	1323.5

Table 7.6 contains the results for series S6 and S7 with a more complex formulation based on Equation (7.2). The stand-alone CPLEX MIP solver was able to solve these problems only in several cases within 5400 seconds, so the exact results are not displayed. The computational time given to the heuristics was 900 seconds for S6 and 3000 seconds for S7. This table indicates that HD is performing similar to MIP-GRASP for what concerns the average and worst case solution quality, but HD outperforms MIP-GRASP concerning the number of best-known solutions it has found.

The GA, however, tends to outperform significantly the two other heuristics on both series S6 and S7. Most likely, this advantage of the GA is due to the effect of the MIP-recombination which constitutes the main difference of the GA from MIP-GRASP. In order to evaluate the significance of combining traits of the parent solutions in the MIP-recombination, we modified the GA so that the full MIP-recombination is performed only with a given probability P_r, otherwise we skip Step 2 in the MIP-recombination routine (this corresponds to mutation without crossover). In Figure 7.2 one can see the frequency of finding the best seen solution as a function of probability P_r for series S3, S5, S6 and S7. This figure indicates that except for the simplest set of instances S3, the capability to combine the features of both parents in a best possible way provides better output of the algorithm. This observation is also supported by the Wilcoxon matched pairs test ($p < 0.1$ for S5, $p < 0.05$ for S6).

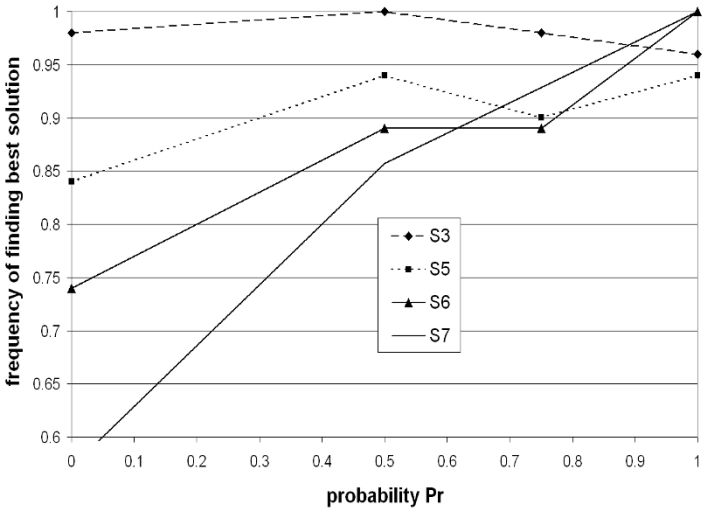

Fig. 7.2 Frequency of finding solutions with best-known objective values as a function of probability P_r.

7.6 Conclusions

In our study, firstly a GRASP approach has been developed in the MIP framework for balancing the transfer lines with multi-spindle transfer machines. The solution construction and local improvement procedures have been implemented in the GAMS environment and the results of computational experiments indicate that the method obtained is quite competitive, especially, for large-scale instances. It is important that due to the flexibility of the MIP modelling tools and robustness of the modern MIP solvers, this approach is applicable to many other large-scale problems, where the straightforward usage of branch-and-cut techniques does not yield satisfactory results.

After that, the research was aimed at the development of a more complex genetic algorithm for the TLBP, using GRASP as a supplementary heuristic for building the initial population. Although for the TLBP instances of small size this approach is not helpful, for the most difficult series of benchmarks, which are not solvable by exact methods in a reasonable amount of time, this method has a significant advantage.

The MIP-based recombination operator is shown to be useful in the genetic algorithm. In view of the wide applicability of general-purpose MIP solvers, we expect that the MIP-recombination approach may be successfully applied for many other problems (see e.g. [3]). However, in the cases where the optimal

recombination can be carried out by polynomial-time algorithms [7], these algorithms should be preferable to general purpose MIP solvers.

In subsequent research it would be valuable to compare the MIP-based approach for GRASP and GA with alternative approaches to metaheuristics, which use the shortest path method in a supplementary graph. Also, it might be helpful to implement grouping of operations into macro-operations in the improvement phase of the MIP-based GRASP. Questions of parameters tuning should be considered as well.

Acknowledgements The authors thank Michael R. Bussieck for helpful discussions on better usage of GAMS potential. The research is supported by the Russian Foundation for Basic Research, grant 07-01-00410.

References

1. I. Baybars. A survey of exact algorithms for the simple assembly line balancing. *Management Science*, 32:909–932, 1986.
2. C. Becker and A. Scholl. A survey on problems and methods in generalized assembly line balancing. *European Journal of Operational Research*, 168:694–715, 2006.
3. P. Borisovsky, A. Dolgui, and A. Eremeev. Genetic algorithms for a supply management problem: MIP-recombination vs. greedy decoder. *European Journal of Operational Research*, 195:770–779, 2009.
4. A. Dolgui, B. Finel, F. Vernadat, N. Guschinsky, and G. Levin. A heuristic approach for transfer lines balancing. *Journal of Intelligent Manufacturing*, 16:159–172, 2005.
5. A. Dolgui, B. Finel, N. Guschinsky, G. Levin, and F. Vernadat. MIP approach to balancing transfer lines with blocks of parallel operations. *IIE Transactions*, 38:869–882, 2006.
6. A. Dolgui, N. Guschinsky, and G. Levin. A special case of transfer lines balancing by graph approach. *European Journal of Operational Research*, 168:732–746, 2006.
7. A. Eremeev. On complexity of optimal recombination for binary representations of solutions. *Evolutionary Computation*, 16:127–147, 2008.
8. P. Festa and M.G.C. Resende. GRASP: An annotated bibliography. In C.C. Ribeiro and P. Hansen, editors, *Essays and surveys on metaheuristics*, pages 325–367. Kluwer, Boston, 2001.
9. S. Ghosh and R. Gagnon. A comprehensive literature review and analysis of the design, balancing and scheduling of assembly lines. *International Journal of Production Research*, 27(4):637–670, 1989.
10. O. Guschinskaya and A. Dolgui. A comparative evaluation of exact and heuristic methods for transfer lines balancing problem. In A. Dolgui, G. Morel, and C. Pereira, editors, *Information Control Problems in Manufacturing 2006: A Proceedings volume from the 12th IFAC International Symposium*, volume 2, pages 395–400. Elsevier, 2006.
11. O. Guschinskaya and A. Dolgui. Balancing transfer lines with multiple-spindle machines using GRASP. Unpublished manuscript, 2007.
12. O. Guschinskaya, A. Dolgui, N. Guschinsky, and G. Levin. A heuristic multi-start decomposition approach for optimal design of serial machining lines. *European Journal of Operational Research*, 189:902–913, 2008.

13. J.P. Hart and A.W. Shogan. Semi-greedy heuristics: An empirical study. *Operations Research Letters*, 6:107–114, 1987.
14. J. Holland. *Adaptation in natural and artificial systems.* University of Michigan Press, 1975.
15. C.R. Reeves. Genetic algorithms for the operations researcher. *INFORMS Journal on Computing*, 9(3):231–250, 1997.
16. M.G.C. Resende and C.C. Ribeiro. Greedy randomized adaptive search procedures. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 219–249. Kluwer Academic Publishers, 2003.
17. A. Scholl. *Balancing and sequencing of assembly lines.* Physica, Heidelberg, 1999.