Abstract
The optimal logical layout design for a type of machining transfer lines is addressed. Such transfer lines are made of many machine-tools (workstations) located in sequence. On each workstation there are several spindle-heads. A spindle-head does not execute a single operation but a block of machining operations simultaneously. Spindle-heads of the same workstation are activated sequentially in a fixed order. The transfer line design problem considered in the paper consists in finding the best partition in blocks and workstations of the set of all operations to be executed on the line. The objective is to minimize the number of spindle-heads and workstations (i.e., transfer line investment cost). An optimal decision must satisfy a desired productivity rate (cycle time) as well as precedence and compatibility constraints for machining operations. A heuristic algorithm is proposed: it is based on the COMSOAL technique and a backtracking approach. Results from computer testing are reported.

Key words: Transfer lines, Line balancing, Optimisation, Heuristic algorithms.
1 Introduction

When mechanical industries want to produce a large quantity of homogeneous products, they often use dedicated transfer lines (Groover 1987, Hitomi 1996). To design such types of machining lines it is necessary to follow three interconnected steps (Dolgui et al. 2006):

- the selection of the necessary operations to machine a part;
- the logical layout design, i.e. the logical synthesis of the manufacturing process, which consists in grouping the operations into blocks (spindle-heads) and the blocks into stations;
- the physical layout and equipment (tools, spindle-heads, etc.) design depending on the logical layout decided.

This paper concerns the logical layout design stage assuming that all the required operations to manufacture a product as well as essential constraints of physical layout are known.

Pieces of equipment involved in such lines represent a significant investment cost. Thus, finding a good (and if possible the best) logical layout design decision that minimises their number is a crucial problem. The aim of our study is to minimise the number of workstations and spindle-heads at the preliminary design stage of the line.

To minimise the number of workstations and spindle-heads (line cost) as well as the occupied floor area for the considered line, the operations are grouped into blocks. All operations of the same block are executed simultaneously by one spindle-head. Thus, the processing time of each block is equal to the longest processing time for all
operations of this block. All blocks (spindle-heads) of the same station are executed sequentially in a given order. The assignment of blocks to a station defines at the same time the order of their activation on this station. Therefore, the station processing time is equal to the sum of block times (for all blocks assigned to this station). All the stations are linearly ordered; there are no buffers between stations. The bottleneck station defines the transfer line cycle time.

In this paper, the objective function is defined as follows:

\[ \text{Minimise } C = mC_1 + QC_2, \quad (1) \]

where

- \( m \) is the number of stations (machine-tools),
- \( Q \) is the total number of blocks (spindle-heads),
- \( C_1 \) and \( C_2 \) are the weighted coefficients (can be interpreted as relative unit costs) for stations and spindle-heads, respectively.

The problem considered in this paper extends the well-known Simple Assembly Line Balancing Problem (SALBP). Some surveys on techniques used to solve SALBP can be found in (Baybars 1986, Boctor 1995, Erel and Sarin 1998, Ghosh and Gagnon 1989, Scholl and Klein 1998, Rekiek et al. 2002). Talbot et al. (1986) present a comparison of heuristic techniques including decisions rules with backtrack and probabilistic search.

The SALBP method cannot be directly used for the considered problem because of the grouping of the operations in blocks where all the operations of the same block are executed simultaneously.
For the investigated problem, two exact optimisation methods based on mixed integer programming (MIP) and graph approaches, respectively, were initially suggested in (Dolgui et al. 1999, Dolgui et al. 2000). An improved version of MIP model with some pre-processing rules and cuts is presented in (Dolgui et al., 2006). These methods can be used only for small and medium-sized problems. For large-scale problems, some heuristic approaches, based on the COMSOAL (Arcus 1996) techniques were developed by Dolgui et al. (2002, 2005). In this paper, one of these heuristics is improved using a backtracking and learning procedure.

The paper is organised as follows. Section 2 introduces some notations and sets the problem statement. In Section 3 a heuristic algorithm is explained. Section 4 introduces a backtracking and learning procedure. In Section 5, numerical examples, tests and comparisons are given. Conclusion remarks are reported in Section 6.

2 Problem Statement

2.1 Notations

The purpose of the line balancing problem addressed herein is to assign the operations to stations (machines) and to blocks (spindle-heads) in such a way that:

- The objective function (1) is optimised;
- The specified line cycle time is not exceeded (i.e., the obtained line cycle time is not greater than the specified one);
- All the constraints for machining operations are satisfied.
The following precedence and compatibility constraints are taken into account:

- A partial order relation on the set of all operations to be machined. This defines a set of possible operations sequences (precedence constraints). In this problem, taking into account the specificity of the machining environment, “an operation \( i \) precedes an operation \( j \)” means that the operation \( i \) can be executed before the operation \( j \) or at the same time than \( j \) (in the same block), but the operation \( j \) cannot be executed before the operation \( i \).

- The necessity to perform some groups of operations on the same station (e.g. because of a required machining tolerance);

- The impossibility to perform some groups of operations within one block (e.g. due to spindle-head design or because of manufacturing incompatibility of operations);

- The impossibility to perform some groups of operations on the same station (e.g. due to station design or because of manufacturing incompatibility of operations).

Figure 1 illustrates the type of lines studied in the paper (www.heller-machinetools.com). The parts are moved from station to station. On each station, there is one or several spindle-heads with different tools. Each tool can execute one or several operations simultaneously on the part. The time spent on a spindle-head is equal or superior to the time of the longest operation executed by this spindle-head. The time spent on a station must be greater (or equal) to the sum of the times spent on the spindle-heads belonging to this station.
Figure 1: A transfer line with multiple spindle-heads

The following notations will be used throughout this paper:

- $\mathbf{N}$ : given set of operations to be machined;
- $T_0$ : desired transfer line cycle time;
- $m_0$ : maximal authorised number of stations;
- $n_0$ : maximal authorised number of blocks in one station;
- $t_j$ : processing time of the operation $j \in \mathbf{N}$;
- $\text{Pred}(j)$ : set of direct predecessors of the operation $j \in \mathbf{N}$;
- $q_0 = m_0 n_0$ : maximal ID of blocks at the line;
- $k$ : station number, $k=1, 2, \ldots, m$;
- $n_k$ : number of blocks at the station $k$;
- $B_q$ : set of operations of the block $q$, i.e. $B_q = N_{kl}$, for $q = (k-1) n_0 + l$;
\( C(k) \) : “cost” of station \( k \) with all its blocks, i.e. \( C(k) = C_1 + C_2n_k \);

\( T_k \) : the slack time of station \( k \);

\( S(k) \) : set of the blocks (their numbers) which will be executed by the station \( k \);

\( \overline{ES} \) : collection of operation sets representing the exclusion (i.e. impossibility) constraints for stations. Operations of the same set of the collection cannot be assigned to the same station all together;

\( \overline{EB} \) : collection of operation sets representing the exclusion (i.e. impossibility) constraints for blocks. Operations of the same set of a collection cannot be assigned to the same block all together;

\( ES \) : collection of operation sets representing the inclusion (i.e. necessity) constraints for the stations. All operations of the same set of the collection must be assigned to the same station.

### 2.2 A MIP Formulation

The problem can be formulated as the following MIP model (Dolgui et al., 2006) with the binary variables:

\[
X_{iq} = \begin{cases} 
1, & \text{if the operation } i \text{ is assigned to the block } q, \\
0, & \text{otherwise}, 
\end{cases}
\]

and the real variables:

\( F_q \geq 0 \), to determine block time, and \( Y_q \geq 0 \) (\( Z_k \geq 0 \)), to count the number of blocks (stations).
Minimise $C_1 \sum_{k=1}^{m_0} Z_k + C_2 \sum_{q=1}^{q_0} Y_q,$ \hspace{1cm} (2)

Subject to:

$$\sum_{q=1}^{q_0} X_{jq} = 1,$$ \hspace{1cm} (3)

$$\sum_{i \in \text{Pred}(j)} \sum_{q' \leq q} X_{iq'} \geq X_{jq} |\text{Pred}(j)|,$$ \hspace{1cm} (4)

$$\sum_{j \in e} X_{jq} \leq |e| - 1, \ e \in EB,$$ \hspace{1cm} (5)

$$\sum_{j \in e} \sum_{q \in S(k)} X_{jq} \leq |e| - 1, \ e \in ES,$$ \hspace{1cm} (6)

$$\sum_{j \in e \setminus \{j(e)\}} \sum_{q \in S(k)} X_{jq} = (|e| - 1) \sum_{q \in S(k)} X_{j(e)q} , e \in ES,$$ \hspace{1cm} (7)

$$F_{q} \geq t_{i} X_{iq},$$ \hspace{1cm} (8)

$$\sum_{q \in S(k)} F_q \leq T_0,$$ \hspace{1cm} (9)

$$Y_{q} \geq X_{jq},$$ \hspace{1cm} (10)

$$Z_k \geq Y_{q}, \ q \in S(k),$$ \hspace{1cm} (11)

$$i,j \in N, \ k=1,...,m_0, \ q=1,...,q_0.$$

The objective function (2) provides the number of stations and blocks minimisation; the constraints (3) require to assign every operation to only one block; the constraints (4) take into account the precedence constraints; the constraints (5) and (6) define the impossibility to include some groups of operations in the same block and at the same station, respectively; the constraints (7) define the necessity to execute some operations at the same station; in the constraints (8) it is assumed that the block time is the longest time of the operations in the block; the constraints (9) provide the given line cycle time;
the constraints (10) and (11) verify the existence of block \( q \) and workstation \( k \) in the
design decision.

2.3 An Example

The problem can be illustrated by the following industrial example. The work-piece to
be manufactured is shown in Figure 2. Table 1 gives the list of the 19 first operations. In
Figure 3, the precedence graph for these operations of the example is given.

![Work-piece to be manufactured](image)

**Figure 2: Work-piece to be manufactured**

<table>
<thead>
<tr>
<th>Nbr</th>
<th>Operation</th>
<th>Nbr</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Drill H1</td>
<td>11</td>
<td>Drill H5</td>
</tr>
<tr>
<td>2</td>
<td>Countersink H1</td>
<td>12</td>
<td>Countersink face of H5</td>
</tr>
<tr>
<td>3</td>
<td>Drill H2</td>
<td>13</td>
<td>Tap H5</td>
</tr>
<tr>
<td>4</td>
<td>Countersink H2</td>
<td>14</td>
<td>Drill H6</td>
</tr>
<tr>
<td>5</td>
<td>Drill H3</td>
<td>15</td>
<td>Countersink face of H6</td>
</tr>
<tr>
<td>6</td>
<td>Countersink face of H3</td>
<td>16</td>
<td>Tap H6</td>
</tr>
<tr>
<td>7</td>
<td>Ream H3</td>
<td>17</td>
<td>Drill H7</td>
</tr>
<tr>
<td>8</td>
<td>Drill H4</td>
<td>18</td>
<td>Countersink face of H7</td>
</tr>
<tr>
<td>9</td>
<td>Countersink face of H4</td>
<td>19</td>
<td>Tap H7</td>
</tr>
<tr>
<td>10</td>
<td>Ream H4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 1: Operations**
The exclusion and inclusion constraints can be expressed as follows:

\[ \overline{ES} = \{\{2, 5\}, \{2, 9\}, \{2, 13\}, \{2, 17\}, \{3, 4\}, \{3, 5\}, \{3, 9\}, \{3, 13\}, \{3, 17\}\}; \]
\[ \overline{EB} = \{\{1, 2\}, \{2, 3\}, \{6, 7\}, \{7, 8\}, \{10, 11\}, \{11, 12\}, \{13, 14\}, \{13, 18\}, \{13, 19\}, \{14, 15\}, \{14, 17\}, \{14, 18\}, \{14, 19\}, \{15, 16\}, \{15, 17\}, \{15, 18\}, \{15, 19\}, \{17, 18\}, \{18, 19\}, \{19, 16\}\}; \]
\[ ES = \{\{7, 11\}\}. \]

In this example, operations 7 and 11 have to be assigned to the same station. Operations 2 and 5 cannot be assigned at the same station. Operations 1 and 2 cannot be assigned to the same spindle-head, etc. An optimal solution for the assignment of operations into blocks of operations and workstations is shown in Figure 4 (eight blocks and four workstations). The arrows show the sequence of blocks and stations.
3. RAP Algorithm

This section explains a heuristic algorithm for transfer line balancing called RAP. This algorithm was suggested in (Dolgui et al., 2002 and Dolgui et al., 2005), it is based on COMSOAL approach (Arcus, 1966).

Remember that the COMSOAL algorithm for SALBP consists of generating random solutions satisfying all constraints, repeating this random generation process a sufficient number of times and keeping the best solution. For the generation of a solution, the operations are assigned one by one (step by step). To do this, a list $L$ of all candidate operations that can be assigned to the current station is created. An operation belongs to this list if all the constraints for this operation are satisfied. At each step, one operation is chosen from this list randomly and it is assigned to the current station. The list is updated after each assignment: the assigned operation and the operations which do not respect the constraints are deleted from the list and the operations for which the constraints become satisfied are added to the list. When the list is empty a new station is created and a new list of candidate operations is built. When all the operations have been assigned, the obtained solution is compared with the best solution to date, the best solution of the two becomes the best current solution, etc.

The COMSOAL is a random search algorithm; it can be illustrated using a decision tree (see Figure 5). At the root node, there is no operation assigned. An arc corresponds to an assignment of one operation, the operation is chosen at random from the list associated with the corresponding father node. One complete branch in this tree gives a solution. To find the next solution (i.e. to make the next iteration), the algorithm goes up
to the root node. The nodes illustrated by a rhombus in Fig. 5 represent the cases when a station is closed (i.e. its maximum number of operations is reached) and a new station is opened.

\[
L_0 = \{i, j, k\}
\]
\[
L_{11} = \{j, p\}
\]
\[
L_{12} = \{i, j\}
\]
\[
L_{13} = \{i, m\}
\]
\[
L_{21} = \{k, m, p\}
\]
\[
L_{22} = \{j, k, m\}
\]

**Figure 5: A decision tree for the COMSOAL method**

The RAP algorithm is inspired from the COMSOAL method, but it takes into account specific constraints and characteristics of the transfer line balancing problem which is more complex: blocks of parallel operations, special type of precedence constraints, inclusion and exclusion constraints, costs of blocks and stations, etc.

Remember that the following notations are used in this paper:

\[
C(k) \text{ is the cost of station } k \text{ including the cost of its blocks;}
\]
\[
T_k \text{ is the slack time of station } k \text{ (see below);}
\]
\[
q_k \text{ is the number of blocks for station } k;
\]
\[
B_q \text{ is the set of operations of block } q.
\]
Like COMSOAL, the RAP algorithm uses a list $L$, which groups all the operations that can be assigned to the current station. For each state of the RAP algorithm, $L$ contains an operation $i$ if:

- The processing time $t_i$ is lower than the slack time $T_k$ of the current station $k$, \textit{i.e.}:
  \[ t_i \leq T_k = T_0 - \sum_{q' \in S(k) \setminus \{q\}} F_{q'}, \]
  where $q$ is the current block ID and $F_q$ is the duration of the block $q$;

- Exclusion constraints $\overline{ES}$ can be satisfied (taking also into account all the operations already assigned to the current station);

- All their predecessors are either already assigned or are in $L$.

One particularity of the considered problem is that the operations linked by precedence constraints can stay together in the list of candidate operations (\textit{i.e.} an operation and its predecessors can be found together in $L$). This is due to the previously mentioned specificity of the precedence constraints and also to the grouping of the operation in blocks.

The RAP algorithm can also be considered as a random search in a decision tree. The root node defines the beginning of the algorithm (no operation is assigned). Each arc means the assignment of one operation to the current block (of the current station). Each node of the enumeration tree is characterised by the set of the operations already assigned. The branching is linked with the choice of one operation to be assigned among all candidate operations from $L$. Each leaf of the tree for which the set of assigned operations is the set $N$ gives a complete solution.
A particularity of RAP is that an additional type of node must be used: the nodes which represent the case when a block is closed. This type of node will be represented by a rectangle. Another specificity is linked to the fact that some iterations can be non conclusive, i.e. a branch must be stopped due to a conflict between some constraints.

When an iteration is finished (conclusive or non conclusive), the next iteration begins from the root node. The non conclusive iteration is ignored.

In other words, if one branch is covered until all the operations are assigned, respecting all the constraints, a solution is found; if no more operation can be added to the branch among non assigned operations, the iteration is non conclusive and a new iteration (from the root of the tree) begins.

For each iteration of the RAP algorithm, one operation is chosen at random in $L$ to be assigned to the current block. If this operation has some predecessors in $L$, then the predecessors are assigned first. When an operation must be assigned, all operations present with this operation in the set $e \in ES$ has to be assigned together.

For each candidate operation, if this operation cannot be assigned to the current block, then a new block is created for the current station. If the slack time or $ES$ constraints prohibit the creation of a new block for the current station, then a new station is created. When a station is closed, the constraints $ES$ are verified. When there is a subset of operations from a set $e \in ES$ not assigned to this station with other operations from the
same set \( e \), then this is a non-feasible solution (the iteration is stopped, it is considered as non conclusive).

When an operation is assigned, \( L \) is updated (the assigned operation is deleted and the operations which become available are added). A conclusive iteration of this algorithm gives a feasible solution for the transfer line.

The RAP algorithm is organised (see Dolgui \textit{et al.} 2005) as one main programme and five procedures (\textit{NewStation}, \textit{NewBlock}, \textit{TryToAssign}, \textit{Assignment} and \textit{AssignToBlock}). The main procedure creates a new station, builds the list \( L \) of candidate operations; then an operation \( \text{Op} \) of \( L \) is selected at random; this operation and all its non assigned predecessors are assigned in a recursive way to build blocks of the current station. If it is not possible, then all the operations of \( L \) that have to be assigned with an operation already assigned (to respect inclusion constraints) are examined: if they cannot be assigned to the current station the iteration stops (it is considered as a non conclusive). If there is no problem, then the criterion \( C \) is increased by the cost of station \( k \) and its blocks, \textit{i.e.} by \( C(k) \). If all the operations of \( N \) have been assigned, then the algorithm stops (\textit{i.e.} a feasible solution has been found); otherwise, a new station is created and the main procedure goes back to the beginning.

The \textit{NewStation} procedure creates a new station (station \( k \)), the station slack time \( T_k \) and the number of blocks of the current station \( q_k \) are initialised by \( T_0 \) and Zero, respectively. The \textit{NewBlock} procedure creates a new block: the duration of the current
block is subtracted from the station slack time, the number of blocks of the current station is increased by one, a new block is created and its duration is initialised by zero.

The procedure \textit{TryToAssign} affects in a recursive way all the predecessors of the operation \textit{Op} (\textit{Op} is the current operation randomly chosen from the list \textit{L}): by searching and treating them as current operation \textit{Op}. The procedure \textit{Assignment} assigns the operation \textit{Op} to the station \textit{k}. The procedure \textit{AssignToBlock} assigns the operation \textit{Op} to the block \textit{q} taking into account exclusion constraints for blocks and stations.

The RAP algorithm gives good results but leads to many non-conclusive iterations (iterations with non-feasible solutions). To increase its performance, a backtracking procedure is developed in the next section.

4. The BAL (Backtracking And Learning) Procedure

4.1 General approach

The RAP algorithm can be interpreted as a random search in the decision tree, so an efficient backtracking procedure can improve its performance. Note that a heuristic of this type for the simple assembly line balancing problem (MALB) has been developed by Dar-El (1973) and showed its performance.

The BAL method will be explained using a decision tree, where a list \textit{L} is associated to each node (this list gives the number of possible branches). The BAL method allows going back in the decision tree from node to another node: this is the backtracking effect. Only nodes which represent closed blocks (no more operation can be added to
these block) are used in backtracking (a step of backtracking consists of going up the previous block).

The branches where good solutions were found are preferred to those where failures (non conclusive iterations or solutions with a bad performance) were got: this is the learning mechanism. The better the solution obtained, the smaller is the probability to cancel a block on this branch in the backtracking procedure.

The following notations are used for the parameters needed to describe the BAL algorithm:

- $Nb$: minimum number of levels (blocks) in the decision tree to backtrack;
- $Random$: a variable uniformly distributed between 0 and 1;
- $Prob$: probability to not backtrack to the precedent level;
- $ProbInit$: algorithm parameter giving the initial value of $Prob$;
- $Nlevel$: counter of the number of backtracked levels;
- $NdblInit$: parameter giving the maximum authorized number of trials to build a block;
- $Ndbl$: counter of the number of trials;
- $CovTree$: parameter allowing the calculation of the maximal number of authorised branching;
- $Nbra(v)$: maximal number of authorised branching from the vertex $v$;
- $Level(v)$: order of the last operation in the assignment sequence from the root to $v$;
\( Weight(i) \) : the probability to be chosen for the operation \( i \);

\( Coef1, Coef2 \) : model parameters.

### 4.2 Some techniques used in BAL

The RAP algorithm is run. Each time a block is closed, a decision is made: either the partial solution is kept and the next block (or station) is opened and the RAP algorithm continues, or a backtracking is executed due to the fact that the end of branch is reached. The end of the branch is reached (\( i.e. \) the current iteration is terminated) if:

- Either, the lower bound (\( LB \)) of the current node is greater than the criterion value for the best solution at this step (upper bound – \( UB \)); it is then sure that, if the search in the branch goes on, the final solution (when all the operations are assigned) will be worse than \( UB \) (the iteration is stopped);
- Or, all the operations are not assigned but no other operation can be add to the current branch while respecting the constraints (the iteration is not conclusive, it is stopped);
- Or, all the operations are assigned, \( i.e. \) a feasible solution is found (the iteration is conclusive).

Each time there is a backtracking and the algorithm restarts from a node, there is a new iteration. In the first case above, the next iteration is started from the root of the decision tree whichever the node where the algorithm stopped (backtracking up to root automatically). For the second and third cases, some specific BAL techniques are proposed. The general schema of algorithm is presented in Figure 6.
4.3 Backtracking

The goal of backtracking is to keep a part of the current solution for the next iteration; because this iteration can go quicker than the one began at the root of the tree: some operations already assigned (using good decisions) have to be kept. The backtracking is done with a probability. The reuse probability should depend on the quality of this branch. The better the solutions on this branch, the sooner it is necessary stopping going up (more blocks of the last decision are involved again). The algorithm climbs up the
current branch block by block (it is not useful to stop at each node but only where a block has been closed). At one node, a decision to stop going up is made: a new branch is started from there.

The \( Nb \) parameter gives the minimum number of levels to go up (minimal number of blocks to cancel), for a complete solution, to continue branching. In fact, it is necessary to go up to a minimum of two blocks for a complete solution to go back to a state from where a different solution than the precedent one can be obtained. The end user must be allowed to set the value of the parameter \( Nb \) by fixing it greater or equal to 2 (i.e. \( Nb \geq 2 \)). Note that this parameter is applied only for complete solution, for a partial solution (i.e. for the case of a non-conclusive iteration), only one block is cancelled.

A value of the variable \( Random \) is generated between 0 and 1. If \( Random \leq (1 - \text{Prob}) \) and if the root has not been reached, the algorithm goes up one level in the tree, new value of \( Random \) is generated and so on, otherwise a new branching starts. The probability \( \text{Prob} \) introduces a random choice to have the possibility to go up one more level. There is only one variable \( \text{Prob} \) in the algorithm because only one branch is treated in the same time. The initial value of \( \text{Prob} \) is given by \( \text{ProbInit} \), which is a parameter of the algorithm. Each new feasible solution obtained by the algorithm changes the current value of \( \text{Prob} \). Better is the obtained solution, higher is the increase of the probability \( \text{Prob} \). Conversely, the worse the current solution, the lower is the new value of the probability \( \text{Prob} \) (see Figure 7). If \( (1 - \text{Prob}) \) is equal to 1 (i.e. \( \text{Prob} = 0 \)), the algorithm goes up automatically (block by block) to the root of the decision tree. At the root, the probability \( \text{Prob} \) takes its initial value \( \text{ProbInit} \).
The way to compute $Prob$ is explained in the subsequent section.

4.4 Stopping conditions

A counter, called $N_{level}$, of the number of levels gone up from the last solutions (number of cancelled blocks) is used. This counter is set to Zero as soon as a new branching is started. It is incremented every time that the algorithm goes up one level (a block has been cancelled).

The stopping conditions for going up are the following:

1) The root of the decision tree has been reached;
2) $N_{level}\geq Nb$ and the current value of $Random > (1-Prob)$.

$Level(v)$ represents the level to which the vertex $v$ belongs, and which is the order number of the last operation in the assigning sequence of operations from the root to $v$. While building a branch, the algorithm goes from the vertex $v$, the level of which is $Level(v)$, to the vertex $v'$, the level of which is $Level(v') = Level(v)+1$ while assigning one operation; for the root vertex $v_0$, $Level(v_0) = 1$. 
For each node $v$ in the decision tree, which represents a closed block, a parameter $Nbra(v)$ is introduced which gives the maximal number of authorised branching from this node. The value of $Nbra(v)$ must depend on the level $Level(v)$, which is the current level in the tree. The closer the algorithm comes to the complete solution (the bigger is the number of assigned operations), the fewer branches can be authorised to be created. The maximal value of $Nbra(v)$ is $|N|$. Of course this parameter is equal to 0 for the leaves of the tree. $Nbra(root)$ is boundless.

To compute $Nbra(v)$, the parameter $CovTree$ is introduced; it corresponds to the rate of desired tree covering. The $CovTree$ value is fixed by the user and depends on the time allocated to the optimisation and on the problem size. It has to be chosen in $]0, 1[$.

For the nodes representing closed blocks, $Nbra(v)$ can be defined as follows:

$$Nbra(v) = \lceil CovTree \times (|N| - Level(v))! \rceil,$$

(12)

The number of branches grows very quickly when $CovTree$ increases. For a node $v$ with $Level(v)=2$ and $CovTree = 1$, $Nbra(v)=(|N| - 2)!$.

If $CovTree = 1/(|N| - w)!$, then for $Level(v)\geq w$, the number of authorised branches is $Nbra(v)=1$, and for $Level(v)<w$, the following formula is obtained:

$$Nbra(v) = \frac{w^{-Niv(v)-1}}{\prod_{l=0}^{w-1} (|N| - Level(v)-l)}.$$

(13)
When a new node representing a block is created, the initial value of $Nbra(v)$ is computed according to formula (13). For each branching from the vertex $v$, the value of the $Nbra(v)$ parameter is decremented as follows: $Nbra(v)=Nbra(v)-1$. When $Nbra(v)=0$, the stopping condition is reached (there is no more possibility to build a new branch from this vertex $v$), the algorithm goes one block up. At the root, $Nbra(v_0)=\infty$.

4.5 Probability to choose an operation

When a node $v$ is reached, the corresponding list of candidate operations $L$ is established from which an operation to be assigned is randomly chosen. Initially, the weight (and in consequence the probability to be chosen) is the same for each operation in $L$: it is equal to 1, i.e. $Weight_i^l=1$, where the superior index gives the current branching number, the inferior index represents the operation number. In other words, the superior index gives the number of times that this node has been found in the current iteration. It grows up any time the algorithm passes again by this node. When the algorithm stops to a node while going up, it does again a random branching. To avoid generating the same branch, the weight of candidate operations is modified. This is made when a block is closed. In this case, the operation which closes the block is known and it is desired to have in the future less chance to choose it, so a smaller value is given to its weight.

$$Weight_i^l = Coef^l \times Prob \times Weight_i^{l-1},$$

where $Coef^l$ is a model parameter $0 \leq Coef^l \leq 1$.

After the modification of the operation $j$ weight, the probability to be chosen for the operation $j$ is normalised and is equal to:
\[ p_j = \frac{\text{Weight}_j}{\sum_{l \in L} \text{Weight}_l}. \]  

(15)

**4.5.1 Non-conclusive iteration treatment.** A particular treatment is added for the non-conclusive iterations. When iteration is non-conclusive (blocking), it is necessary to go one block up and to change the weight of all the candidate operations from \( L \). The weight of all the operations of the last closed block (and not only of one operation as before) must be modified as follows:

\[ \text{Weight}^l_i = \text{Coef}2 \times \text{Weight}^{l-1}_i, \]

(16)

where \( \text{Weight}^l_i \) is the weight of the operation \( i \) for the \( l \)-th trial to obtain a conclusive iteration, \( \text{Weight}^1_i = 1 \), \( \text{Coef}2 \) is a coefficient (parameter) of the model, \( 0 \leq \text{Coef}2 \leq 1 \); \( i \in L \). The obtained weights of operations are copied in all the descending vertexes until a no new station is created (for which all the constraints are respected).

To avoid an infinite loop, a new parameter \( \text{NdblInit} \) and a counter \( \text{Ndbl} \) are introduced. \( \text{NdblInit} \) gives a maximum number of trials. The counter \( \text{Ndbl} \) is set to zero, when a failure (non-conclusive) is established. It is incremented at each new trial. When the counter \( \text{Ndbl} \) is equal to \( \text{NdblInit} \), the trials are stopped and the algorithm begins to go up in the tree following the procedure of the previous section.

**4.5.2 Learning (computing of the probability \( \text{Prob} \)).** The learning technique is based on the calculation of the probability \( \text{Prob} \) characterising the branch quality. If \( \text{Prob} \) is equal to 1, then the algorithm stops, an optimal solution is founded. If \( \text{Prob} \) is lower
than one, then the algorithm goes block by block up in the decision tree, each time with
the probability \(1-Prob\). Consequently, \(Prob\) must significantly reduce if the iteration is
of bad quality or stopped, or in the case of blocking (no-conclusive iteration); it must
increase in case of good iteration. The positive increasing must be proportional to the
solution quality.

To compute \(Prob\), the upper bounds (initial \(UB_0\) and running \(UB\)) and lower bounds
(initial \(LB_0\) and running local \(LLB\)) of the objective function are used. The difference
between the initial bounds \(UB_0\) and \(LB_0\) corresponds to an estimation of the maximum
distance \(\text{DstMax}\) between the solutions:

\[
\text{DstMax} = UB_0 - LB_0.
\] (17)

The initial upper bound \(UB_0\) is equal to:

\[
UB_0 = C_1 m_0 + C_2 q_0.
\] (18)

The initial lower bound \(LB_0\) is equal to:

\[
LB_0 = C_1 m + C_2 n = C_1 \max_{i \in \mathbb{N}} (k^- [i]) + C_2 \max_{i \in \mathbb{N}} (q^- [i] | n_0=1, \text{ES} = \{\emptyset\}).
\] (19)

where \(k^- [i]\) and \(q^- [i]\) are the lower bounds of possible indices for workstation and
blocks, respectively, where the operation \(i\) can be assigned (see in Dolgui et al. 2000 the
corresponding pre-processing procedure tacking into account all the problem constraints
and the methods to compute \(k^- [i], q^- [i], k^+ [i]\) and \(q^+ [i]\)). For example, if an operation \(i\)
cannot be in the same block than the operation \(j\) and it precedes the operation \(j\), then if
the smallest block where \( i \) can be assigned is equal to \( q[i] \) then the smallest block where \( j \) can be assigned is \( q^*[j] > q^*[i] \).

If a branch is not running to the end (all the operations are not already assigned), then there is a partial solution \( p \) with \( N \subseteq \mathbb{N} \) assigned operations. For this solution, the criterion \( C(p) \) is equal to:

\[
C(p) = C_1 \cdot m(p) + C_2 \cdot n(p),
\]

where \( m(p) \) and \( n(p) \) are the number of stations and the number of blocks, respectively.

To obtain a local lower bound, \( LLB \) for \( p \), after having assigned a set of operations \( N \), it is necessary to fix \( q^-[p][j] = q^+[p][j] = q[j] \) and \( k^-[p][j] = k^+[p][j] = k[j] \), for \( j \in N \) and to computed again \( q^+[p][i] \) and \( k^-[p][i] \) for \( i \in N \setminus N \) using the same algorithm (see Dolgui et al. 2000). Here, \( q_p[j] \) and \( k_p[j] \) are the numbers of blocks and stations for the operation \( j \) in the partial solution \( p \) and where \( k^-[p][i] \) and \( q^-[p][i] \) are the lower bounds of possible indices for workstation and blocks, respectively, where the operation \( i \) is in the partial solution \( p \).

\[
LLB = C_1 \max_{i \in N} (k^-[p][i]) + C_2 \max_{i \in N} (q^-[p][i]) \mid n_0=1, ES = \{\emptyset\}, \quad (20)
\]

When all the operations of \( N \) are assigned, there is a complete feasible solution \( P \):

\[
C(P) = C_1 \cdot m + C_2 \cdot n.
\]

As soon as the new value of the criterion is better than the current upper bound \( UB \), it becomes the new current \( UB \).
There are four different cases for probability $Prob$ calculation:

1. If $LLB \geq UB$, i.e. the local lower bound for the partial solution $p$ is greater or equal to the best complete solution found then the iteration is stopped, $Prob = 0$ ($1 - Prob = 1$), the algorithm goes block by block up to the root.

2. If it is no more possibility to assign operations due to the constraints (a non-conclusive iteration), the value of $Prob$ is corrected while decreasing it in such manner:

$$Prob_s = Prob_{s-1} - Prob_{s-1} \left( 1 - \frac{UB - LLB}{UB_0 - LB_0} \right). \quad (21)$$

3. If a feasible solution is found but without getting a better value for the current upper bound, then:

$$Prob_s = Prob_{s-1} - Prob_{s-1} \left( \frac{C(P) - UB}{UB_0 - UB} \right). \quad (22)$$

4. If a good solution is found (all the operations are assigned and the criterion is better than current $UB$), then:

$$Prob_s = Prob_{s-1} + (1 - Prob_{s-1}) \frac{UB - C(P)}{UB - LB_0}. \quad (23)$$

If $C(P) = LB_0$, $Prob = 1$ ($1 - Prob = 0$), then the algorithm stops, the optimal solution is found.

5. Some test results

Tests were carried out on a Compaq W6000 bi-processor machine, Intel Xeon with 1.70 Ghz for CPU and 1 Giga RAM.
Table 2 gives the computation results for the set of examples tested for RAP in (Dolgui et al., 2005). The first column gives the number of operations, the second and third ones give the constraints on the maximal number of blocks for one station and on the maximal number of stations allowed. Then, five columns show the used parameters. The next two columns give the criterion value obtained with RAP and BAL, and the last two columns present CPU times.

<table>
<thead>
<tr>
<th>NO</th>
<th>n₀</th>
<th>m₀</th>
<th>Parameters</th>
<th>Criterion Value</th>
<th>CPU Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>1/ CovTree Coef Coef NdblInit ProbInit</td>
<td>RAP RAP + BAL</td>
<td>RAP RAP + BAL</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>6</td>
<td>360 0.75 0.35 6</td>
<td>0.65 26 (3,4) 26 (2,3)</td>
<td>1&quot; 0.08&quot;</td>
</tr>
<tr>
<td>13</td>
<td>6</td>
<td>3</td>
<td>660 0.75 0.35 6</td>
<td>0.65 36 (3,3) 36 (3,3)</td>
<td>1&quot; 0.13&quot;</td>
</tr>
<tr>
<td>15</td>
<td>6</td>
<td>3</td>
<td>24024 0.7 0.5 6</td>
<td>0.5 36 (3,3) 36 (3,3)</td>
<td>55&quot; 1.17&quot;</td>
</tr>
<tr>
<td>35</td>
<td>3</td>
<td>7</td>
<td>33390720 0.7 0.5 6</td>
<td>0.5 40 (4,6) 40 (3,5)</td>
<td>6&quot; 0.83&quot;</td>
</tr>
<tr>
<td>45</td>
<td>3</td>
<td>6</td>
<td>130320960 0.7 0.4 9</td>
<td>0.7 54 (4,7) 42 (3,6)</td>
<td>1'04&quot; 11.5&quot;</td>
</tr>
</tbody>
</table>

Table 2. Parameters and results for RAP and RAP+BAL

The parameter values have been chosen in the following manner: several runs of the algorithm are made. For each run, ten different values of a parameter are given (fixing the others). After these runs, the values which gave the best results for each parameter are kept and then the algorithm is run again.

Four simple industrial examples have also been tested (see Table 3). The first line represents the example shown in this paper (see Section 2.1). The utilisation of BAL
with RAP allows us to reach very quickly the optimal value for the criterion (this optimal value is obtained also using MIP model with Cplex solver). The criterion value is optimal and it was reached for both methods.

| $n_0$ | $m_0$ | Criterion Value | CPU Time | RAP | RAP+BAL |
|---|---|---|---|---|
| 2 | 6 | 56 (4,8) | 62" | 0.31" |
| 2 | 6 | 42 (3,6) | 3" | 1.02" |
| 9 | 3 | 30 (2,5) | 3" | 0.4" |
| 8 | 3 | 44 (3,7) | 5" | 0.51" |

**Table 3. Comparison for CPU times (industrial examples)**

Tableau 4 shows the results for an example with 100 operations. Algorithms have been run from 10000 to 60000 with a step of 10000 iterations. The first column gives the number of iterations; the two following columns present the two criterion values (one for each algorithm) then follow the CPU times.

<table>
<thead>
<tr>
<th>Number of iterations</th>
<th>Criterion</th>
<th>CPU Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAP</td>
<td>RAP+BAL</td>
<td>RAP</td>
</tr>
<tr>
<td>6000</td>
<td>82</td>
<td>82</td>
</tr>
<tr>
<td>10000</td>
<td>82</td>
<td>68</td>
</tr>
<tr>
<td>20000</td>
<td>82</td>
<td>68</td>
</tr>
<tr>
<td>60000</td>
<td>82</td>
<td>68</td>
</tr>
</tbody>
</table>

**Table 4. Comparison RAP and RAP+BAL for example with 100 operations**
6. Conclusion

A crucial step of machining line design was considered. It concerns the logical layout design, i.e. the logical synthesis of the manufacturing process, which consists in grouping the operations into blocks (spindle-heads) and the blocks into stations. The error at this step (or an inefficient solution) increase drastically the line and machining product costs. Therefore, the development of decision support methods and tools for logical layout design of this type of machining line is a challenging perspective.

An original approach was suggested in this paper. It deals with mathematical modelling of the considered problem as a mixed integer program and with the search a good and if possible optimal solution of this model minimizing the number pieces of equipment (spindle heads, stations) and consecutively the line cost.

The model which was suggested is a new line balancing model, more complex that all known models, because of the parallel execution of operations, and several specific compatibility and precedence constraints of machining environment. The exact solution of the suggested model is possible only for small and medium sized industrial problems. For large scale problems, a solution is to develop some dedicated heuristics taking into account the structure of constraints and the specificity of the problem.

A heuristic algorithm to find a “good” design decision has been also presented in this paper. This heuristic is an extension the COMSOAL based method RAP. For develop this extension, the problem has been presented as a problem of random search in a
decision tree. The corresponding algorithm to explore this decision tree has been proposed. It used several techniques of backtracking and learning. These techniques were developed in details taking into account the specific constraints of the considered problem.

The results of tests are promising. The employment of the proposed techniques improves drastically the heuristic RAP, decreasing the CPU times and giving better results from the criterion value point of view for the problems tested.

The perspectives of our research consist of an analysis of the suggested algorithm parameters. These parameters have to be adjusted and tested to reach better results. It is also interesting to produce more test results to study the behaviour of the developed techniques and to find new ways to increase the algorithm calculation efficiency and results.

Acknowledgements
The authors thank Dr. N. Guschinsky and Pr. G. Levin for discussions, help and data for the industrial cases of this problem.

References


Ghosh, S., and Gagnon, R. A comprehensive literature review and analysis of the
design, balancing and scheduling of assembly lines. *International Journal


Martin, P. and D'Acunto, A., Design of a production system: an application of

Rekiek, B., Dolgui, A., Delchambre, A., and Bratcu, A. State of the art of assembly

Scholl, A. and Klein, R., Balancing assembly lines effectively: a computational

Talbot, F.B., Patterson, J.H., Gehrlein, W.V., A comparative evaluation of heuristic line