Università degli Studi di Roma Tre
Dipartimento di Informatica e Automazione
Via della Vasca Navale, 79 – 00146 Roma, Italy

Loading parts and tools in a Flexible Manufacturing System

Dario Pacciarelli

RT-DIA-38-1998 **June 1998**

ABSTRACT

We consider the problem of assigning parts and tools on a Flexible Manufacturing System composed by W identical parallel workstations, so that the workload of the workstations is well balanced. The goal is to minimize the total number of tools needed on all workstations, including multiple copies, if any. We give an integer programming formulation of this problem and a strong cutting plane algorithm to solve it. We used cover inequalities for the $0-1$ knapsack constraints and other valid inequalities to strengthen the formulation and applied branch and cut method. Computational experiences on some real world problems and randomly generated problems are reported.

1 Introduction

Automated production planning algorithms can be used to improve productivity in various manufacturing environments. This paper considers one such environment, a Flexible Manufacturing System (FMS) composed by four identical parallel workstations. Each workstation can perform different operations, provided that it is equipped with the required tools.

In a common decomposition strategy (see, for example, [2]) the production plan at a mid-term (or tactical) level is obtained by solving three strongly inter-related problems: the part type selection (i.e. how many part types, and how many parts of each type, should be processed within a given planning horizon, typically a shift), the part routing (i.e. on which workstation to produce each part), and the tooling (i.e. find the best allocation of tools to workstations in order to process the selected parts).

Part type selection is a well known problem in production planning and has been studied by many authors. In a rigid approach part types are partitioned into batches (see for example [14, 6, 9]). In such an approach all parts in a batch are machined continuously until all requirements for all parts are finished.

A different approach to part type selection is referred to as flexible approach [11]. In flexible approach part types in a batch are machined in certain relative ratios that optimize particular indexes, such as workload balance. A flexible approach to part type selection usually yields better system operation performance than a rigid one.

The definition of the tooling problem may vary depending on many factors, and in particular on the chosen approach to the part type selection. Two main cases can be distinguished:

1. Static tool allocation: in this case tools cannot be moved from one workstation to another while the workstation is running. Re-tooling is permitted only at the beginning of each shift, or when an emergency occurs, such as the failure of a tool.

2. Dynamic tool allocation: in this case workstations can share tools via a tool handling system.

The influence of tool management on the overall performance of automated production systems has been pointed out by several authors (comprehensive surveys on this topic can be found in [5, 12]).

At the individual machine level, tool management subsumes the problem of allocating tools to the machine and simultaneously sequencing the parts to be processed so as to minimize some measure of the production performance.

In multi-machine environments (see, for example [3, 10]) the problem consists of partitioning parts and tools required by the selected part types among the workstations so that the workload of the workstations is well balanced and the number of tools optimizes particular indexes, such as the number of duplication (static tool allocation) or the number of tool changes (dynamic tool allocation). In [3] the static tool allocation problem is formulated as a non-linear mixed-integer program, and it is solved by several linearizations. In [13] the same problem is formulated as a 0-1 integer program and it is solved by Lagrangian relaxation. Other authors (see, for example [4, 7]) proposed heuristic methods to solve static as well as dynamic tool allocation problems.

Dynamic tool allocation allows lower tooling costs, despite a more complex tool management policy. Static tool allocation requires higher tooling costs, because of the multiple copies required for many tools. On the other hand, it requires very simple tool management policy and allows better performances in terms of throughput and system reliability, due to the fact that each workstation can work independently from the others.

In this paper flexible part type selection and static tool allocation are analyzed: the goal is to minimize tool duplication. In section 2 the production environment is summarized. A description of the optimization model is given in section 3. In section 4 computational results are reported.

2 The System

The manufacturing system considered is installed into an avionics components industry. It is a common FMS composed by four parallel workstations, equipped with a large on-board tool magazine with 120 available positions. Workstations are connected by a rail guided vehicle in charge of moving parts between them.

Material handling system comprises two rotary stations used for fixturing and load/unload operations. A big buffer can host parts (raw products, sub-processed and finished products), pallets, fixtures as well as fixtured pallets with the raw parts to be processed. Parts and tools transportation system are fully automated.

The orders are released weekly and partitioned among shifts. Each part to be processed has to undergo a known set of operations, or *part program*, to be executed consecutively. Parts requiring the same part program are said to belong to the same *part type*. We define a *lot* as the set of all parts in a part type. The production plan in a shift consists of N lots of parts to be processed. Once loaded on a workstation, a part is not removed until the completion of its part program, i.e. preemption is not allowed, due to high quality standard requirements. Let n_i denote the number of parts in lot i. The execution of a part program for a part of lot i requires a known *processing time* p_i. Let $q_i = n_i p_i$ be the time needed to process all parts in lot i.

Each operation in a part program requires a specific *tool* which must be present at the spindle of the machine when the operation starts. Therefore, each part requires a set of tools for its production. Let T_i be the set of tools required for executing the part program for a part of lot i.

The tool magazines local to the workstations must contain all the tools needed in each shift. The automated tool transportation system is to be used as a setup device and to perform consumed tool substitutions. Hence, if two part programs requiring the same tool must be executed on two different workstations, then the required tool must be *duplicated*, i.e. it must be loaded on the on-board tool magazine of both workstations, even if the two part programs are executed at different times. In other words the supervisory controller of the FMS is unable to schedule tools usage during a shift. Let T be the set of all different tools needed to process all parts of all lots in a shift, i.e. $T = \bigcup_{i=1}^{N} T_i$.

3 The loading problem

The loading problem requires to assign jobs and tools to the workstations within a given time horizon, typically a shift. Even if in our application the number of workstations

equals four, in this section we concern with a general number W of workstations. Decision variables, parameters and symbols used in the integer programming formulation are listed below.

- $x_{jk} = 1$ if the tool k is assigned to the workstation j, 0 otherwise,
- f_{ij}: fraction of lot i to be processed on the workstation j,
- T_i: set of tools required by each part of lot i,
- T: set of all tools required in the shift, i.e. $T = \bigcup_{i=1}^{N} T_i$,
- W: number of the workstations,
- N: number of part types to be processed in the shift.

Defining q_i as the time consuming of all parts of part type i in the shift, and c_{jk} as the cost of assigning tool k to workstation j, we obtain the following mixed linear program. Of course, the case $c_{jk} = 1$ corresponds to minimize tool duplications. Let us denote this problem as $(P1)$.

Problem (P1):

$$\min \quad \sum_{j=1}^{W} \sum_{k \in T} c_{jk} x_{jk} \tag{1}$$

$$\sum_{j=1}^{W} f_{ij} = 1 \qquad i = 1, \ldots, N \tag{2}$$

$$\sum_{i=1}^{N} q_i f_{ij} = \frac{1}{W} \sum_{i=1}^{N} q_i \qquad j = 1, \ldots, W \tag{3}$$

$$x_{jk} - f_{ij} \geq 0 \qquad i = 1, \ldots, N; \quad j = 1, \ldots, W; k \in T_i \tag{4}$$

$$f_{ij} \geq 0 \qquad i = 1, \ldots, N; \quad j = 1, \ldots, W \tag{5}$$

$$x_{jk} \in \{0, 1\} \qquad j = 1, \ldots, W; k \in T \tag{6}$$

Inequalities (2) ensure each lot to be completely processed. Inequalities (3) provide the workload balance among workstations. Inequalities (4) ensure the presence, on each workstation, of all tools needed by the part types assigned to that workstation.

Notice that a feasible solution always exists and it corresponds to trivially assign $f_{ij} = \frac{1}{W}$ for all i, j and $x_{jk} = 1$, for all j, k. Note also that, by fixing the variables x_{jk} to 1 or to 0, the problem becomes an assignment problem, and therefore it can be solved in polynomial time.

In order to obtain an initial integer solution, better than the trivial one, we apply the procedure shown in Figure 1.

Our branch and bound algorithm for (P1) is based on the Linear Programming relaxation obtained from model (1)-(6) by substituting constraints (6) with the following:

$$0 \leq x_{jk} \leq 1. \tag{7}$$

Let us denote by (P1R) the LP relaxation of (P1) obtained from (1)-(5) and (7). A generic step in the branch and bound algorithm requires to chose an open problem and

Procedure ISTL (Initial Solution Tool Loading)
Input:PLI formulation of the problem.
Output: Feasible solution vectors $\tilde{x}$.
begin

for $j = 1, \ldots, W$, $\forall k \in T$, initialize variables $\tilde{x}_{jk} = 1$, i.e. assign all tools to all workstations;

set exists= true;

while exists

remove a tool from a workstation, i.e. set $\tilde{x}_{jk} = 0$ in such a way that the remaining assignment problem has a feasible solution, if no tool can be removed, set exists = false;

end

end

Figure 1: Procedure ISTL

to solve its LP relaxation: if the problem is unfeasible or its relaxed solution is integer, the problem is closed, otherwise a branch strategy must be adopted and one or more open problems must be generated. Any time an integer solution is found it is compared with the current optimum. If it is better it becomes the current optimum. If the relaxed solution of an open problem gives an optimal value greater than or equal to the current optimum, the problem is closed and no others open problems are generated from that open problem.

The algorithm terminates when no more open problems exist. In order to avoid not acceptable time consuming in finding an optimal solution we interrupt the algorithm processing when the maximum time consuming available is reached. The current optimal solution is assumed to be the optimal one.

For each step in the branch and bound algorithm we chose the open problem with minimum LP relaxation value. The following branch step is adopted:

1. chose an index i such that $\forall j = 1, ..., W$ a variable x_{jk}, $k \in T_i$, exists that is fractional; if not such an index exists, go to step 3.

2. Construct W sub-problems P_j , $j = 1, ..., W$, by setting in the problem $P_j : x_{jk} = 1, \forall k \in T_i$. STOP

3. If the search at step 1 does not find any index i, choose the fractional variable x_{jk} such that $c_{jk}(1 - x_{jk})$ is maximum and construct two sub-problems by setting $x_{jk} = 1$, and $x_{jk} = 0$ respectively. STOP

The effectiveness of the branch and bound algorithm greatly depends upon the *tightness* of the LP relaxation solved at each step of the branch and bound. Let us formally define the tightness of an LP relaxation. Let us consider two different mixed integer formulations A and B of the same problem P. Let P_A and P_B denote the set of feasible

solution of the LP relaxation of A and B respectively. We say that formulation A is tighter than B if $P_A \subset P_B$.

Looking for tight formulations helps in solving branch and bound algorithms. In fact, since an open problem is closed every time its LP relaxation gives an optimal value integer or greater than or equal to the current (integer) optimum, one is interested in finding an LP relaxation such that its optimal value is as close as possible to the integer solution of the problem, i.e. a relaxation as tight as possible. In this perspective, we look for new valid constraints to be added to the model (1)-(6), which are redundant as long as binary constraints (6) are imposed, but capable of improving the value of the LP relaxation. In this way we obtain a new linear relaxation of the problem (P1), tighter than (P1R).

We will consider a special class of such additional constraints. This class derives from the Knapsack substructure of (P1), and contains *cover* inequalities (see, e.g., [8]). Let consider one of the inequalities (3). Variables f_{ij} are not binary, nonetheless, due to constraints (4), they have the binary variables x_{jk} as upper bound. Therefore we can write the following valid Knapsack inequalities:

$$\sum_{i=1}^{N} q_i x_{jk(i)} \geq \frac{1}{W} \sum_{i=1}^{N} q_i \tag{8}$$

where $k(i)$ is any index $k \in T_i$, associated to the real variable f_{ij}, for each $i = 1, ..., N$. Since the objective of the problem lead to minimize the number of variables $x_{jk} = 1$, we would like to set to zero as many variables as possible. In fact, every time a variable x_{jk} is set to zero, all parts in any lot i which need tool k must be processed on some machines different from j. Since, from equation (3), the maximum workload of all machines different from j is $\frac{W-1}{W} \sum_{i=1}^{N} q_i$, then the maximum number of variables x_{jk}, $k \in T$, we can set to zero is strictly lesser than T, for a given j. In order to introduce the *cover* inequalities we need to change the variables as follows. Let us introduce the new variables $y_{jk} \in \{0, 1\}$, defined as:

$$y_{jk} = 1 - x_{jk}. \tag{9}$$

From (8) and (9), we have:

$$\sum_{i=1}^{N} q_i y_{k(i)} \leq \frac{W-1}{W} \sum_{i=1}^{N} q_i \tag{10}$$

Since some of the variables $y_{jk(i)}$ may refer, in general, to the same variable, we will consider in the following a general constraint in the form:

$$\sum_{k \in T} a_k y_{jk} \leq \frac{W-1}{W} \sum_{i=1}^{N} q_i = b \tag{11}$$

A cover C of equation (11) is a subset of T such that $\sum_{k \in C} a_k > b$. A cover is said to be *minimal* if $\sum_{k \in C - \{h\}} a_k \leq b$ for any $h \in C$. For each cover C we can write the following inequality, which is valid for all the integer solutions of (11):

$$\sum_{k \in C} y_{jk} \leq |C| - 1 \tag{12}$$

The above class of additional constraints (12) contains a huge number of members, and even the number of different equations of type (8) grows exponentially with $|T|$ and N.

Hence, embedding them all explicitly in the model is impractical even for small values of $|T|$ and N. On the other hand, since we are interested to obtain a tighter formulation for (P1) for a particular numerical instance, we only need a few of them. Thus one can use the iterative scheme shown in Figure 2, where the additional constraints (12) are added run-time to the current LP relaxation.

Procedure ADDCUTS
begin

set up the initial LP relaxation (P1R) with constraints (1)-(5) and (7) only;

repeat

solve the current LP model, and let x^* be its optimal solution;

change variable space to y by applying equations (9);

if there exist inequalities (12) which are violated by x^* then add them all to the current LP relaxation;

until no violated inequality has been found;

end

Figure 2: Procedure ADDCUTS

In this way, at every iteration we add to the current LP relaxation only those constraints which cut the current optimal solution of the LP relaxation of the problem. The key point of the above scheme is the identification of violated constraints belonging to class (12). Since the exhaustive enumeration is impractical, due to the exponential number of such inequalities, a more sophisticated identification procedure for this class is needed, which we briefly outline in the sequel (for the details we refer to [8]). We will look for the most violated cover inequality among those, if any, able to cut the optimal solution of the linear relaxation of (P1). This can be done by solving the following knapsack problem:

$$\max \quad \sum_{k\in T}(y^*_{jk}-1)z_{jk} \qquad \begin{cases} \sum_{k\in T} a_{jk}z_{jk} \geq \lfloor b+1 \rfloor \\ z_j \in \{0,1\}^n \end{cases} \tag{13}$$

where y^*_{jk} is the optimal solution of the linear relaxation of (P1), and z_j is a binary vector such that $k \in C$ if and only if $z_{jk} = 1$. If we find a cover inequality able to cut the solution $y^*_{jk} = 0$, we can apply equations (9) to obtain the new valid inequality in the original space x:

$$\sum_{k\in C} x_{jk} \geq 1. \tag{14}$$

Notice that, in the above knapsack problem, each variable z_{jk} has a non-positive 'cost' $(y^*_{jk} - 1)$ and a nonnegative 'weight' a_{jk}. Therefore we can fix the z-variables associated to the $0-1$ valued variables in the y^*_{jk} as follows:

- fix $z_{jk} = 1$ for all indices h with $y^*_{jk} = 1$, since it does not change the objective function, while it increases the left-hand side of the constraints in (13).
- fix $z_{jk} = 0$ for all indices h with $y^*_{jk} = 0$, since otherwise the resulting cover would not violate the LP optimum y^*.

As a result of the above variable-fixing, we are allowed to solve a 'restricted' knapsack problem. Moreover, we do not need to solve the knapsack problem at optimality. In fact, in order to cut the relaxed optimal solution y^*, we only have to find a solution $\tilde{z}$ such that $\sum_{k \in T}(y^*_{jk} - 1)\tilde{z}_{jk} > -1$. Therefore, we heuristically solve the restricted knapsack problem with considerable saving of computing time required to determine a new valid inequality.

It is also possible to *lift* the cover inequalities to obtain stronger valid inequalities (and therefore a tighter formulation). In fact, suppose an index $h \notin C$ is found, such that the inequality $\sum_{k \in C \cup \{h\}} x_{jk} \geq 2$ is valid for Problem (P1), then a stronger valid inequality has been found. We do this by the procedure shown in Figure 3.

Procedure Lifting
Input: a_{jk}, b, C.
Output: new cover C'.
begin

set $A = \{a_{jk} : k \in C\}$, $B = \{a_{jk} : k \notin C\}$, $C' = C$;

repeat

$a'_{jk} = \max\{a_{jk} : a_{jk} \in B\}$;
Let d be the sum of the $|C|$ smallest elements of $A \cup a'_{jk}$.
if $d > b$ **then** $A = A \cup a'_{jk}$, $B = B - \{a'_{jk}\}$, $C' = C' \cup \{k\}$;

until $d > b$

end

Figure 3: Procedure Lifting

At this point we have a good procedure to determine a cover inequality, if it exists, starting from a particular knapsack inequality in the form (8). Since there are an exponential number of such inequalities, coming from a single equation (3), we need a procedure to determine run-time a set of promising inequalities of the form (8) in order to formulate and solve problem (13). For each $j = 1, \ldots, W$ we chose the *most promising* inequality as follows. The key observation is that it is more convenient to associate to the largest variable y^*_{jk} the largest a_{jk} as possible. Therefore, for each machine j, we apply the procedure in Figure 4.

Adding cover inequalities to the LP formulation helps in solving the branch and bound since it reduces substantially the total number of open problems to analyze. In table 1

Procedure Select Knapsack
begin

for $j = 1$ to W do

for $i = 1$ to N do

set $y_{jk(i)} = y_{jh}$ such that $y_{jh} = \max\{y^*_{jk} : k \in T_i\}$

end

Figure 4: Procedure Select Knapsack

Table 1: Performance of the branch and bound codes.

Size	nodes		constraints	
N/T	BB	BB1	BB	BB1
25730	126	38	879	910
25/50	4108	3802	857	910
25/130	5316	5024	879	899
30/80	68614	62844	1026	1046
30/130	35186	27120	1062	1200

we evaluate the performances of procedure ADDCUTS. The two branch and bound algorithms considered here (BB and $BB1$) only differs in the usage of the covers inequalities. BB is the simple branch and bound code, while $BB1$ is the code with the procedure $ADDCUTS$. Table 1 shows the effectiveness of this procedure, especially for large-scale instances, i.e. when the maximum number of open problem is a critical factor, due to the limited amount of memory available on the computer. In the first column the size of the considered instance is reported. The first number indicates the total number of jobs N in the instance, the second number is the total number of tools $|T|$ in the instance, while the total number of workstations has be fixed equal to $W = 2$ in all instances. Second and third column report the total number of LP problems solved with BB and BB1 code, respectively. In the fourth and fifth column the total number of constraints is reported, respectively in the original formulation of the problem and while solving the problem with BB1.

4 Numerical results

The production scenery was simulated on the basis of the production requirements of a typical Flexible Manufacturing System produced by an Italian producer of FMSs. The system is installed into a French avionics component industry.

The daily production scenery considered is composed by 20-30 different part types with a total number of 100-150 different tools. Each part type requires from 15 up to 20

Table 2: Numerical Results.

Instance	solution value			time (sec)	
Size	ISTL	SPLIT	Opt	SPLIT	B & B
6/20	39	34	33	0.56	1615
8/25	42	40	39	1.86	16152
15/60	119	108	108	8.99	$>$ 20 h
20/100	248	224	223	304.8	$>$ 20 h
25/100	250	219	219	701.2	$>$ 20 h
25/120	234	220	218	1000.6	$>$ 20 h
30/120	329	302	301	9438.8	$>$ 20 h
30/150	377	337	335	19190	$>$ 20 h

tools and the processing time is from 30 up to 500 minutes. Our attempts to solve exactly real size problems were unsuccessful for problems with $W \geq 3$. However, the algorithm is very efficient in solving 2-machines problems. Therefore, a system with $W = 4$ machines has been considered for the experiments. For such a system, an heuristic (referred to in the table as $SPLIT$) has been used. $SPLIT$ is based on a first step in which a two macro-machines partition is performed and a second step in which each macro-machine is divided into two machines. In Table 2 computational results are shown. In column 1 the number of part types and tools for each instance are shown. In column 2, 3 and 4 the solution of the ISTL heuristic, SPLIT heuristic and the optimum value are given respectively. In column 5 the time consuming of split heuristic is given, while in column 6 The time consuming of the branch and bound code is reported. The maximum computing time allowed for computation was fixed equal to 20 hours. When this time limit was reached we stopped the computation and reported in the fourth column of Table 2 the best integer solution found within the time limit. Observe that the improvement to heuristic SPLIT after 20 hours computing time is limited to one or two tools. All programs are in C language, and they make use of CPLEX for solving the LP relaxation. The timings reported are computed on a RISC 6000-3AT.

References

References

[1] A. Agnetis, C. Arbib and K.E. Stecke, *Part type selection and optimal scheduling in a flexible flow system*, Proceedings of the Second International Conference on CIM, Troy, NY 1990.

[2] R.G. Askin, and C.R. Stanridge, *Modeling and analysis of manufacturing systems* (John Wiley & Sons, 1992).

[3] M. Berrada and K.E. Stecke, A branch and bound approach for machine load balancing in flexible manufacturing systems, Management Science 32, 1316–1335 (1986).

[4] J.P. Follonier, *On grouping parts and tools and balancing the workload in a flexible manufacturing systems*, Research Report ORWP 92/05, Ecole Politechnique fédérale de Lausanne, Département de mathématiques, Lausanne, Switzerland (1992).

[5] A.E. Gray, A. Seidmann and K.E. Stecke, *A synthesis of decision models for tool management in automated manufacturing*, Management Science 39, 549–567 (1993).

[6] S. Hwang, *A constraint-directed method to solve the part type selection problem in flexible manufacturing systems planning stage*, in: FMS, Operations research Models and applications, K.E. Stecke and R. Suri (editors) (Elsevier, Amsterdam, The Netherlands, 297–309, 1986).

[7] R. Macchiaroli and S. Riemma, *a modified C-meac clustering algorithm to minimize THS utilization and conflicts on tool usage in a two machines flexible manufacturing cell* proceedings of the 1994 Japan-U.S.A. symposium on flexible automation, 111–118 (1994).

[8] G.L. Nemhauser and L.A. Wolsey, *Integer and combinatorial optimization* (John Wiley & Sons, New York).

[9] S. Rajagopalan, *Formulation and heuristic solutions for parts grouping and tool loading in flexible manufacturing systems*, in: FMS, Operations research Models and applications, K.E. Stecke and R. Suri (editors) (Elsevier, Amsterdam, The Netherlands, 311–320, 1986).

[10] M.S. Sodhi, A. Agnetis, and R.G. Askin, *Tool addition strategies for flexible manufacturing systems*, The International Journal of Flexible Manufacturing Systems 6, 287–310 (1994).

[11] K.E. Stecke and I. Kim, *A flexible approach to implementing the short term FMS planning function*, in: FMS, Operations research Models and applications, K.E. Stecke and R. Suri (editors) (Elsevier, Amsterdam, The Netherlands, 283–295, 1986).

[12] D. Veeramani, D.M. Upton, and M.M. Barash, *Cutting-tool management in computer integrated manufacturing*, The International Journal of Flexible Manufacturing Systems 3/4, 237–265 (1992).

[13] J.A. Ventura, F.F. Chen, and Wu Chih-Hang,*Grouping parts and tools in flexible manufacturing systems production planning*, International Journal of Production Research 28, 1039–1056 (1990).

[14] C.K. Whitney and T.S. Gaul, *Sequential decision procedures for batching and balancing in FMSs*, Annals of Operations Research 3, 301–316 1985.