12-1-1996

# Heuristic Solutions for Loading in Flexible Manufacturing Systems

Bharatendu Srivastava
*Marquette University*, bharat.srivastava@marquette.edu

Wun-Hwa Chen
*Washington State University*

Published version. *IEEE Transactions on Robotics and Automation*, Volume 12, No. 6 (December 1996), DOI. 

# Heuristic Solutions for Loading in Flexible Manufacturing Systems

Bharatendu Srivastava and Wun-Hwa Chen

*Abstract*—**Production planning in flexible manufacturing system deals with the efficient organization of the production resources in order to meet a given production schedule. It is a complex problem and typically leads to several hierarchical subproblems that need to be solved sequentially or simultaneously. Loading is one of the planning subproblems that has to addressed. It involves assigning the necessary operations and tools among the various machines in some optimal fashion to achieve the production of all selected part types. In this paper, we first formulate the loading problem as a 0-1 mixed integer program and then propose heuristic procedures based on Lagrangian relaxation and tabu search to solve the problem. Computational results are presented for all the algorithms and finally, conclusions drawn based on the results are discussed.**

## I. Introduction

A FLEXIBLE manufacturing system (FMS) is an integrated manufacturing facility that consists of a group of workstations linked with an automated material handling system to move the parts and working under the direction of a central computer. Many automation concepts and modern technologies are incorporated into the system, such as numerically control (NC) machine tools, computer numerically control (CNC) machine tools, and direct numerically control (DNC) machine tools, robots, automated material handling system and automated inspection using vision systems or pressure-sensitive sensors. Currently, FMS's exist in a variety of configurations, degree of complexity, and in a wide range of capacities (for a classification of FMS see [31]). They are capable of producing a variety of parts at varying production rates, batch sizes, and product mix. Although they are used for producing parts as diverse as small precision components used in instrumentation to very large structural components for construction equipment, yet their main applications have been in metal cutting, forming, and assembly operations for small to mid-volume batch production in automobiles, electrical equipment, machinery, and aerospace industries [33]. Some of the new FMS's consist of very versatile machines having a high degree of reliability and can be run unattended [23]. To date, hundreds of these systems have been installed all over the world. An FMS has the potential to offer several strategic and operational benefits over conventional manufacturing systems (such as reduction in manufacturing lead times, and an ability to respond quickly and effectively to disturbances in the production system or changes in the marketplace). However, its efficient management requires solution to complex and difficult production planning problems. In this paper, we focus on loading one of the problems in production planning for FMS.

Production planning in FMS's is concerned with the organization of production resources to satisfy a given master production schedule. The objective of this activity is to develop an acceptable cost effective feasible production plan over the planning horizon. Due to the size and complexity of most realistic FMS planning problems and its related computational difficulties, the planning problem is generally decomposed hierarchically into several smaller and tractable problems. Decisions made at the higher levels in the decomposition schemes have significant impact on lower level decisions, in the sense that they limit the range and scope of the lower level decisions. One of the most comprehensive and extensively cited decomposition of the FMS planning problem was given by Stecke [41], in which she suggested the following five subproblems: i) part type selection which determines a subset from the set of part types having production requirements for immediate and simultaneous processing, ii) machine grouping which partitions the set of machines so that each machine in a particular group can perform the same set of operations, iii) production ratio which determines the relative proportions in which selected part types will be produced, iv) resource allocation which allocates the limited number of pallets and fixtures of each fixture type to the selected part types, and v) loading which allocates operations and required tools for the selected part types among the machine groups subject to technological and capacity constraints of the FMS. Jaikumar and Van Wassenhove [23] proposed a three-level hierarchy with the top level selecting the part types and the associated production quantities, the next level assigns the selected part types and the available tools among the various machines with the last level dealing with the part-machine scheduling. There have been other hierarchical decomposition schemes also, such as those proposed by Kusiak [28], Buzacott [6], and Van Looveren, Gelders, and Van Wassenhove [45]. Even though these decomposition schemes are different, yet they share many similarities. Generally, the subproblems generated from such a hierarchical decomposition scheme are solved either simultaneously or sequentially or in an iterative manner. Further, the interdependency between the subproblems must be maintained throughout or it can lead to infeasibilities at a later stage. The focus of this paper is to address one of

Manuscript received August 29, 1994; revised August 21, 1995. This paper was recommended for publication by Editors P. B. Luh and A. Desrochers upon evaluation of reviewers' comments.

B. Srivastava is with the Department of Management, Marquette University, Milwaukee, WI 53201-1881 USA.

W.-H. Chen was with the Department of Management and Systems, Washington State University, Pullman, WA 99163 USA. He is now with the Department of Business Administration, National Taiwan University, Taipei, Taiwan.

Publisher Item Identifier S 1042-296X(96)07249-7.

the planning subproblems, namely the loading problem and to develop efficient solution approaches for solving it. Loading in FMS is considered to be one of the most important planning subproblems [30], [28] for which efficient algorithms are needed [43].

The loading problem consists of assigning the various operations and tools among the various machines, to achieve the production of all selected parts according to one or more objective and subject to certain limitations. It specifies the specific tools to be loaded in each machine's tool magazine and the machines to which the operations should be routed. The exploitation of the inherent flexibility of the manufacturing system in terms of operation assignment is crucial to the achievement of a good utilization of resources. In this regard, several important attributes and limitations of a FMS need to be considered in developing an effective loading strategy. Included in this list are the major characteristics of the manufacturing system such as an ability to process an operation on any of the several available machines, operations could then require different processing time and cutting tools on different machines, tools could also be shared amongst the different operations, limited capacity of the tool magazines, the availability of the number of copies of cutting tools, and the amount of processing time available at each machine [43]. The tool magazine capacity limits the number of tools that can be loaded on the machine. It can also constrain the number of different tool types that can be loaded, since certain tool types often need specific slot positions. Moreover, there are situations where the actual number of tool slots occupied by certain tools depends on the actual placement of the tools in the magazine. Complicating the situation further is the multicriteria nature of this problem, and in the past several objectives have been suggested by many researchers [28], [41], [43] such as minimize the part movement between various machines, balance the workload among the machines, minimize production costs, duplicate some operations, and maximize system throughput. Thus, loading in FMS is a difficult problem to solve as the above description of the problem leads to a nonlinear optimization problem [41], which is difficult to solve to optimality even for a reasonably sized practical problem. Hence, a complete and precise formulation of this problem is most likely to be computationally complex. However, in practice tool magazine capacities and available processing times are considered to be the most important constraints [27], while balancing the workload and maximizing the throughput are often the desired objectives [39].

Loading in FMS's has been considered by several researchers in the past (for example see [5], [27], [28], [36], [41]) who have proposed different models and methods for various configurations of FMS production systems. Stecke [41] formulated the loading problem as a nonlinear integer program, which was solved using linearization methods. For the same formulation, a branch and bound solution approach was given by Berrada and Stecke [5], and a two-stage branch and backtrack procedure was suggested in [39], based on the approach outlined in [5]. A bicriterion objective of balancing the workloads among the machines and meeting the job due dates for a random FMS was considered in [38]. Stecke and Talbot [43] developed several heuristic procedures based on some of the well known approximate algorithms for the bin packing problem. Kouvelis and Lee [27] avoid nonlinearity (due to tool magazine capacity) in their model, by appropriately defining the "operations" and "tool types" and reformulate the problem to develop an efficient branch and bound algorithm which exploits the block angular structure of the reformulated model. Kusiak [28] developed four different loading formulations on the assumption that tools are not shared among the operations (thereby avoiding the nonlinear tool magazine capacity constraint) and showed the relationship of these models to two of the well-known problems, the generalized assignment and the transportation problem. Machining cost resulting from various tool-machine combinations was considered by Sarin and Chen [37] in their 0-1 integer programming formulation. However, their model becomes difficult to solve as the problem size increases. Kim and Yano [24] view the loading problem as a two-dimensional bin packing problem (tool slots and processing time) and/or scheduling tasks on uniform processors (processors having different speeds, but the speed is independent of the task under execution) to minimize makespan. Savings due to tool commonality were explicitly considered in their algorithms. It is important to mention that in the past, many researchers have integrated the loading problem with other planning subproblems such as part type selection [7], [35], [30], machine grouping [42], tool allocation [37], and scheduling. Liang and Dutta [30] considered the part selection, loading and tool configuration for a class of FMS's with no tool transportation mechanism. They proposed a sequential-bicriteria approach involving Lagrangian heuristic.

In this paper, we present a Lagrangian relaxation based heuristic and two tabu search based heuristics as solution techniques for the loading problem. The paper is organized as follows. In Section II, the mathematical formulation of the loading problem is presented, Section III describes the three solution procedures developed to solve the problem. In Section IV, we present the computational results, and finally some concluding remarks are given in Section V.

## II. Loading Problem Formulation

In this section, we present a 0-1 mixed integer program formulation of the loading problem, which minimizes the makespan, the maximum completion time of all operations (i.e., the processing time on the most heavily loaded machine type). We first give a description of the manufacturing facility considered and then state and justify the assumptions behind the mathematical formulation.

The FMS production facility considered here consists of a set of machine types denoted by the set $J$, capable of performing all the different operations in the set $I$, which need to be carried out for the selected part types scheduled for production in the upcoming period of the planning horizon. Each machine type $j \in J$ consists of one or more identical machines. Further, each machine within a machine type has a known tool slot capacity of its tool magazine denoted by $b_j$. We also assume that machines within a machine type are identically tooled so that they can individually perform the same set of operations. Therefore, only one machine of each type needs to

be considered for the number of tool slots needed to load the necessary tools in its tool magazine to carry out an operation. It is important to note that identical machines can be partitioned into several machine types and then loaded with different set of tools. For example, there are FMS's that consists of all general purpose machines where the functionality of a machine is determined solely by the set of tools loaded in their tool magazine. Hwan and Shogan [21] have considered such an FMS in the context of part type selection problem. For each operation $i \in I$, there is a set of machine types $J_i \subset J$, capable of performing operation $i$, and there is a corresponding set of operations $I_j \subset I$, that machine type $j$ can perform. An operation $i \in I$ can be performed on any of the several alternative machines types $j \in J_i$ with varying degree of efficiency measured in terms of processing time $t_{ij}$, and different cutting tools and therefore, the number of tool slots $a_{ij}$ needed. Further, the processing time requirement $t_{ij}$, takes into consideration the number of machines within each machine type (split equally among them). This is because of the following fact, processing time for an operation is computed by multiplying the unit operation processing time of the selected part type on the machine type with the associated number of units of that part type scheduled for production in that batch. Thus, splitting the operation time equally amongst the machines within a machine type amounts to dividing the number of units of the selected part type equally amongst the available machines. If for some technical or other reason, the processing of an operation cannot be split amongst the different machines of a machine type, then $t_{ij}$ denotes the processing time of an operation $i$ on *one* of the machines of machine type $j$. The existence of alternative machine types for an operation is because of a partial overlap in terms of operations the machine types can perform. This is especially evident in industries where the process technology is evolving continuously [29]. The kind of production system considered here is similar to the multi-cell flexible manufacturing system of MacCarthy and Liu [31], and to the generic description of an FMS given by Jaikumar and Van Wassenhove [23] which according to them is in widespread use.

We assume that a batch of part types and their associated production quantities have already been determined for immediate processing. In terms of Stecke's [41] decomposition scheme these two problems correspond to part type selection and production ratio. The selection of part types into *batches* is generally based on profit margins, due dates, processing time requirements, or some other appropriately defined criterion. For example, if meeting due dates is very important to a company, then part type selection is carried out in such a way that most of the due date requirements are satisfied. In fact, due dates is the most common criteria used in part type selection and a number of mathematical models have been proposed for it (see [21], [25]). Detailed sequencing and scheduling follows the loading problem [23], [28], [45]. With this approach, the part types are partitioned into batches and once the processing of a batch commences, there is no preemption, i.e., the batch has to be processed completely. On completion of the batch, the machines are set up for the next batch and the tools that are no longer needed are replaced with tools required for the next batch.

Once the part types have been selected for processing, the next step in the planning process is to carry out an assignment of the necessary operations and tools amongst the various machines, i.e., the loading problem. Because of batching, one can assume that all selected part types for the next batch will be available for processing at time zero. With this background, the loading problem can be formulated as the following 0-1 mixed integer program:

$$\mathbf{P}: \quad v(P) = \min T \tag{1}$$

$$\text{s.t.} \quad \sum_{i \in I_j} t_{ij} x_{ij} \leq T \quad \forall j \tag{2}$$

$$\sum_{j \in J_i} x_{ij} = 1 \quad \forall i \tag{3}$$

$$\sum_{i \in I_j} a_{ij} x_{ij} \leq b_j \quad \forall j \tag{4}$$

$$x_{ij} = 0 \text{ or } 1 \quad \forall j J_i, \quad \forall i \in I \tag{5}$$

$$T \geq 0 \tag{6}$$

where

$x_{ij}$ is a 0-1 decision variable that has value 1 if operation $i$ is assigned to machine type $j$, and 0 otherwise,

$T$ is the makespan to be minimized.

The objective function (1) and constraint (2) define the makespan $T$, the maximum completion time of all the operations. In other words, this objective minimizes the total production time, which is an effective surrogate for maximizing the system productivity and utilization of the machines. Minimizing makespan is one of the most important criterion [8], especially when dealing with numerically controlled (NC) machines. In the face of increasing global competition, many firms using FMS are adopting the Just-in-Time manufacturing philosophy, and minimizing makespan in such an environment will lead to reductions in manufacturing lead times and safety stocks for downstream operations. This objective function has also been used by Flanders and Davis [10] in scheduling FMS at Caterpillar, Inc., and by Kim and Yano [24]. Thus, with the above loading objective, one can effectively carry out the actual sequencing and scheduling of the operations to reduce the machine idle time. Constraint (3) specifies that each operation can be assigned to only one machine type. Constraint (4) is the tool magazine capacity limitation. Integrality and nonnegativity of the decision variables are imposed by constraints (5) and (6), respectively. The above formulation balances the workload among the different machine types by exploiting the inherent flexibility of the system in terms of operation assignment.

Our model does not consider tool overlap, as mathematical consideration of all possible overlap amongst the various operations can result in an *enormous* number of nonlinear terms even for a relatively small problem size (see [38], [41] for details). Confounding this problem further, is the requirement that the saving in tool slots depend upon the physical placement of the tools in the tool magazine. Thus, consideration of tool overlap will result in a complex nonlinear mixed integer program which is quite difficult to solve to optimality [4]. In all likelihood, the problem would become

difficult from a computational perspective, which may limit its utility from a practical standpoint. On the other hand, ignoring tool overlap provides for redundancy for certain tool types, which can be useful during tool breakage, a major cause of quality problems in FMS [23]. A growing number of new FMS are now being run unattended for one or two shifts a day and multiple copies of some tool types is essential, if the useful life of the tools is less than the unattended period of operation. In addition, disregarding tool overlap adds to flexibility in assigning multiple copies of tools to balance the workload among the machines at scheduling stage [21].

## III. Heuristic Solutions

In this section we propose three algorithms to solve the problem **P**. These algorithms are a Lagrangian based heuristic [9], and two tabu search based heuristics [12]. Since the solution obtained from these heuristics may not be optimal, a reassignment procedure (or a slight variant of it) is used by all three heuristics to improve upon it. This procedure is described next, as it will be referred to in subsequent discussion.

### A. Procedure REASSIGNMENT

This procedure accepts any feasible solution as input and attempts to improve upon it (by reducing its makespan) by reassigning some of the operations from the machine type having the largest total processing time to other machine types capable of performing the relevant operations. It is an adaptation of an improvement scheme described in [16] and is outlined next. Let $I_j^A$ be the set of operations currently assigned to machine type $j$, then the complete REASSIGNMENT can be described as

Step 0: Initialize. Let $T_j = \Sigma_{i\in I_j^A} t_{ij}$ and $b_j = b_j - \Sigma_{i\in I_j^A} a_{ij}, \quad \forall j \in J.$

Step 1: Find $j_o \in J$ such that $T_{j_o} = \max_{j\in J}\{T_j\}$.

Step 2: Find any operation $i \in I_{j_o}^A$ and machine type $j \in J_i, j \neq j_o$, such that $T_j + t_{ij} < T_{j_o}$ and $a_{ij} \leq b_j$.

Step 3: If step 2 was successful then reassign operation $i$ from machine type $j_o$ to $j$,
$T_{j_o} = T_{j_o} - t_{ij_o}; \quad T_j = T_j + t_{ij};$
$I_{j_o}^A = I_{j_o}^A \backslash \{i\}; \quad I_j^A = I_j^A \cup \{i\};$
$b_{j_o} = b_{j_o} + a_{ij_o}; \quad b_j = b_j - a_{ij};$
Go to step 1
Else stop (no more improvement possible).

It is important to note that this is just an improvement scheme and the quality of the final solution depends upon the initial feasible solution.

### B. Heuristic 1

Lagrangian relaxation and subgradient optimization will be used to develop an effective heuristic (Lagrangian heuristic) for the solution to problem **P**. Lagrangian relaxation has been successfully applied to several combinatorial optimization problems (see for example [9], [11]), after Held and Karp [17], developed a highly successful algorithm for the traveling salesman problem using this technique. In general, many hard combinatorial problems can be viewed as having a special structure (easy to solve) complicated by a set of difficult constraints. A relaxation of such a problem produces a Lagrangian problem which capitalizes on the special structure of the problem, by moving the set of difficult constraints into the objective function through a vector of Lagrangian multipliers. The resulting Lagrangian problem is solved using an efficient and specialized algorithm. The optimal value of the Lagrangian problem bounds the original objective value (in our case a lower bound on $v(P)$). Therefore, to develop such a heuristic, we relax constraint sets (2) and (3), with multipliers $\mu \in \Re_+^{|J|}$ and $\pi \in \Re^{|I|}$ respectively. The reason for choosing this way of relaxing **P**, is that the resulting problem then separates into $|J|$ knapsack problems (to be shown later), which can then be easily solved to optimality using a pseudopolynomial algorithm [34]. We thus have:

$$\mathbf{LRP_{\mu,\pi}}: \quad v(LRP_{\mu,\pi}) = \min T\left(1 - \sum_{j\in J}\mu_j\right) + \sum_{j\in J}\sum_{i\in I_j}(\mu_j t_{ij} - \pi_i)x_{ij} + \sum_{i\in I}\pi_i$$
$$\text{s.t.} \quad (4), (5), \text{ and } (6). \tag{7}$$

From the Lagrangian duality theory for integer programming [11], we have,

$$v(LRP_{\mu,\pi}) \leq v(P), \qquad \mu_j \geq 0 \quad \forall j \in J,$$

i.e., $v(LRP_{\mu,\pi})$ is a lower bound on $v(P)$, and the best choice of $\mu$ and $\pi$ can be obtained by solving the Lagrangian dual with respect to constraint (2) and (3), i.e.,

$$\mathbf{LD}: \quad v(LD) = \max v(LRP_{\mu,\pi}) \tag{8}$$
$$\text{s.t.} \quad \mu_j \geq 0 \quad \forall j \in J. \tag{9}$$

As can be seen there is only a nonnegativity restriction on $T$ in $\mathbf{LRP_{\mu,\pi}}$, and if $\Sigma_{j\in J}\ \mu_j > 1$, then $T$ is unbounded from above and $\mathbf{LRP_{\mu,\pi}}$ is unbounded, and therefore, $v(LRP_{\mu,\pi})$ is $-\infty$. This is because $\mathbf{LRP_{\mu,\pi}}$ is a minimization problem and the coefficient of $T$ in (7) becomes negative. On the other hand, if $\Sigma_{j\in J}\ \mu_j \leq 1$, then the optimal value of $T$ in the relaxed problem $\mathbf{LRP_{\mu,\pi}}$ will be zero and $\mathbf{LRP_{\mu,\pi}}$ then decomposes into $\mathbf{LRPj_{\mu,\pi}}$ for each machine type $j$,

$$\mathbf{LRPj_{\mu,\pi}}: \quad v(LRPj_{\mu,\pi}) = \min \sum_{i\in I_j}(\mu_j t_{ij} - \pi_i)x_{ij} \tag{10}$$
$$\text{s.t.} \quad \sum_{i\in I_j} a_{ij}x_{ij} \leq b_j \tag{11}$$
$$x_{ij} = 0 \text{ or } 1 \quad \forall i \in I_j. \tag{12}$$

Thus, the Lagrangian relaxation problem $\mathbf{LRP_{\mu,\pi}}$ reduces to $\mathbf{LRPj_{\mu,\pi}}$, a single constraint 0-1 knapsack problem for each machine type $j$, which can then be easily solved by dynamic programming or branch and bound [34]. In fact, we can replace $\Sigma_{j\in J}\ \mu_j \leq 1$ with $\Sigma_{j\in J}\ \mu_j = 1$, without affecting $v(LD)$, because of the following fact. If $\Sigma_{j\in J}\ \mu_j \leq 1$, then in $\mathbf{LRP_{\mu,\pi}}$, $T$ should be chosen as small as possible and since there is only a nonnegativity constraint on $T$, the optimal value

of $T$ is zero, and therefore, the coefficient of $\mu_j$ in **LD** (a maximization problem) is $\Sigma_{i\in I_j}\ t_{ij}$, which is nonnegative. (However, for a given solution, the actual makespan in **P** can be computed by finding $\max_{j\in J}\{\Sigma_{i\in I_j}\ t_{ij}x_{ij}\}$. Therefore, to obtain the optimal value of the Lagrangian relaxation dual **LD**, one should restrict the multipliers $\mu_j$ to be nonnegative and $\Sigma_{j\in J}\ \mu_j = 1$ (a projection method is used to enforce this restriction and is described later on).

Since $v(LRP_{\mu,\pi})$ is concave, and piecewise linear, an important issue in Lagrangian relaxation is the magnitude of the duality gap i.e., $v(P) - v(LD)$. Generally, when some variables are required to be integral there is a duality gap present most of the times, which implies $v(LD) < v(P)$. Moreover, when a duality gap is present, we do not achieve the global optimum of the original problem. However, $v(LD)$ does provide a bound on $v(P)$ at least as strong as the LP-relaxation of **P**, since $\mathbf{LRP}_{\boldsymbol{\mu},\boldsymbol{\pi}}$ does not possess the integrality property as defined in [11]. The integrality property indicates that the optimal value of $\mathbf{LRP}_{\boldsymbol{\mu},\boldsymbol{\pi}}$ is not altered by dropping the integral restriction on its variables. Determination of a good set of multipliers plays a pivotal role in deriving bounds by Lagrangian relaxation. Subgradient optimization [18] and other multiplier adjustment methods [3], have proven to be efficient methods of updating the multipliers. Multiplier adjustment methods are heuristics that effectively utilize the special structure of a particular problem, whereas the subgradient method is a fairly standard approach for solving the dual problem. We use the subgradient optimization technique to solve **LD**. An overview and theoretical convergence properties of the subgradient optimization algorithm is given in [18].

Starting with an initial set of multipliers, each iteration of the subgradient optimization procedure requires solving the Lagrangian relaxation $\mathbf{LRP}_{\boldsymbol{\mu},\boldsymbol{\pi}}$ which involves solving $|J|$ single constraint 0-1 knapsack problems each having at most $|I|$ variables, and then updating the multipliers. It is important to note that the subgradient direction is not necessarily an ascent direction, therefore, $v(LD)$ may not increase at each iteration, however, if the step size $\theta$ (a scalar that specifies the extent of movement in the gradient direction) is sufficiently small the new solution obtained is closer to the optimal one. After each update of the Lagrangian multipliers, a projection is made to enforce $\Sigma_{j\in J}\ \mu_j = 1$. Held, Wolfe and Crowder [18] have designed a very simple procedure to accomplish this projection, which is outlined next.

Let $\overline{\mu} = \mu + \theta\Gamma$, where $\theta$ and $\Gamma$ be the step size and the subgradient vector respectively, then for a given $\overline{\mu} \in \Re_+^{|J|}$ we need to find a vector $\mu$, such that $|\mu - \overline{\mu}|^2$ is minimum and satisfies the condition $\Sigma_{j\in J}\ \mu_j = 1$, which mathematically involves solving the following problem:

$$\min\left\{\tfrac{1}{2}|\mu-\overline{\mu}|^2\colon \sum_{j\in J}\mu_j = 1,\ \text{and}\ \mu_j \geq 0 \quad \forall j\in J\right\}.$$

Let the coordinates of $\overline{\mu}$ be ordered such that $\overline{\mu}_1 \leq \overline{\mu}_2 \leq \cdots \leq \overline{\mu}_{|J|}$, then in the subgradient procedure, $\mu$ is modified as

$$\mu_j = \max(\overline{\mu}_j - \beta, 0)$$

where

$$\beta = \frac{-1 + \sum_{j=j_o+1}^{|J|}\overline{\mu}_j}{|J| - j_o}$$

and

$$j_o = \max\left\{j\colon \frac{\overline{\mu}_{j+1} + \cdots + \overline{\mu}_{|J|} - 1}{|J| - j} > \overline{\mu}_j\right\}.$$

Let $\mu^0$ and $\pi^0$ denote the initial set of multipliers. Initially, we set $\mu_j^0 = 1/|J| \quad \forall j \in J$, and since $t_{ij} \geq 0$ for all $(ij)$, the solution $x = 0$ is optimal in $\mathbf{LRP}_{\boldsymbol{\mu},\boldsymbol{\pi}}$ for any $\pi_i \leq \mu_j t_{ij}$, for all $(ij)$, therefore, a good choice for $\pi^0$ is, $\pi_i^0 = \min_{j\in J_i}\{\mu_j^0 t_{ij}\} \quad \forall i \in I$. An attempt is made to generate a feasible solution to problem **P** at every iteration of the subgradient optimization algorithm. Also, it is easy to see that $T$ is bounded from above by $\max_{j\in J}\{\Sigma_{i\in I_j}\ t_{ij}\}$. The complete subgradient procedure for solving problem **P** is summarized below:

Step 0: Initialize. $k = 0$; (iteration counter)
$\lambda^k = 2$;
Let $\mu_j^k = 1/|J| \quad \forall j \in J$.
Let $\pi_i^k = \min_{j\in J_i}\{\mu_j^k t_{ij}\} \quad \forall i \in I$.
$v(P)^{UB} = \max_{j\in J}\{\Sigma_{i\in I_j}\ t_{ij}\}$.
$v(P)^{LB} = 0$.

Step 1: Solve $\mathbf{LRP}_{\mu^k,\pi^k}$ for a given $\mu^k$ and $\pi^k$, for each machine type $j \in J$ and obtain $v(LRP_{\mu^k,\pi^k})$. This gives us the lower bound $v(P)^{LB}$. Update the lower bound and $\lambda^k$, if necessary.

Step 2: If $v(P)^{UB} = v(P)^{LB}$, then stop; we have an optimal solution.

Step 3: Compute the subgradient vectors $\Phi^k$ and $\Gamma^k$ as follows:
$\Phi_i^k = 1 - \Sigma_{j\in J_i}\ x_{ij} \quad \forall i \in I$,
$\Gamma_j^k = \Sigma_{i\in I_j}\ t_{ij}x_{ij} \quad \forall j \in J$.

Step 4: If $\Phi_i^k = 0 \quad \forall i \in I$, then go to step 5. Otherwise, apply procedure RESTOREFEASIBILITY, in an attempt to generate a feasible solution for **P**, based on the current solution. Update $v(P)^{UB}$ accordingly.

Step 5: Compute the step size $\theta$ by,
$\theta^k = \lambda^k(v(P)^{UB} - v(LRP_{\mu^k,\pi^k}))/\|\Phi^k\|^2 + \|\Gamma^k\|^2$.

Step 6: Update $\pi$ and $\mu$ as,
$\pi_i^{k+1} = \pi_i^k + \theta^k\Phi_i^k \quad \forall i \in I$,
$\mu_j^{k+1} = \max(\mu_j^k + \theta^k\Gamma_j^k - \beta^k, 0) \quad \forall j \in J$.

Step 7: If the iteration count has exceeded the limit, then go to step 8. Otherwise set $k = k + 1$ go to step 1.

Step 8: Apply procedure REASSIGNMENT to improve the best feasible solution obtained for **P**.

We start with an initial $\lambda^0 = 2$ and halve it, whenever the lower bound has failed to increase within 15 consecutive iterations. The algorithm is terminated after 100 iterations or when the difference between the feasible solution value and the lower bound is less than 1 (applies only when all $t_{ij}$'s are

integers). This step size reduction does not guarantee global convergence, but performs well with respect to the bound obtained and the computational effort required. Additionally, if the performance after several iterations of the subgradient optimization is not satisfactory, a branch and bound embedding Lagrangian relaxation will be necessary.

### C. Procedure RESTOREFEASIBILITY

At each iteration of the subgradient optimization algorithm, the solution is either feasible with respect to the original problem **P** or infeasible. Infeasibility may be due to multiple assignments for some operations or operations with no assignment. Procedure RESTOREFEASIBILITY is a greedy approach and is an attempt to restore feasibility with respect to problem **P**. Since none of the tool magazine constraints are violated, restoring feasibility amounts to eliminating multiple assignments and assigning the unassigned operations to the machine types with available tool magazine capacity. Our computational experience shows that the number of operations violating constraint (3) is small and feasibility can be restored most of the time using this procedure. Since this approach takes only a minimal effort, it is used at every step of the subgradient optimization method and is described next.

Let $I' = \{i \in I: \Sigma_{j \in J_i} x_{ij} > 1\}$ be the set of operations with multiple assignments and $I^0 = \{i \in I: \Sigma_{j \in J_i} x_{ij} = 0\}$ be the set of operations with no assignment. Further let $I_j^A = \{i \in I: x_{ij} = 1\} \quad \forall j \in J$, procedure RESTOREFEASIBILITY first removes all operations with multiple assignments, in a way that attempts to reduce the current makespan by the maximum, and then assigns sequentially the unassigned operations to a machine type (with enough available tool slots), that results in a minimal increase in the makespan

Step 0: Initialize. Let $T_j = \Sigma_{i \in I_j^A} t_{ij}$ and $b_j = b_j - \Sigma_{i \in I_j^A} a_{ij} \quad \forall j \in J$.

Step 1: If $I' = \emptyset$, then go to step 4 (no multiple assignments), Else let $T_{j(1)} \geq T_{j(2)} \geq \cdots \geq T_{j(|J|)}$ and $k = 1$.

Step 2: If $I' \cap I_{j(k)}^A \neq \emptyset$, then find $i_o \in I' \cap I_{j(k)}^A$, such that $t_{i_o j(k)} = \max_{i \in I' \cap I_{j(k)}^A} \{t_{ij_{(k)}}\}$. Else $k = k + 1$, and go to step 2.

Step 3: Remove operation $i_o$ from machine type $j_{(k)}$, and update $I'$, if necessary $I_{j(k)}^A = I_{j(k)}^A \backslash \{i_o\}; \quad b_{j(k)} = b_{j(k)} + a_{i_o j(k)};$ Go to step 1.

Step 4: If $I^0 = \emptyset$, then stop (feasibility restored). Else let $J_i^C = \{j \in J_i: a_{ij} \leq b_j\} \quad \forall i \in I^0$. If $J_i^C = \emptyset$ for any $i \in I^0$, then stop (no feasible solution generated).

Step 5: Find $j(i) \in J_i^C$, such that $T_{j(i)} + t_{ij(i)} = \min_{j \in J_i^C} \{T_j + t_{ij}\} \quad \forall i \in I^0$.

Step 6: Find $i_o \in I^0$, such that $T_{j(i_o)} + t_{i_o j(i_o)} = \min_{i \in I^0} \{T_{j(i)} + t_{ij(i)}\}$.

Step 7: Assign operation $i_o$ to machine type $j(i_o)$, and update $I^0 = I^0 \backslash \{i_o\}; \quad b_{j(i_o)} = b_{j(i_o)} - a_{i_o j(i_o)};$ Go to step 4.

### D. Heuristic 2

Heuristic 2 is based on the tabu search (TS) method developed by Glover [12], [13]. It is an adaptive procedure and can be superimposed on any local improvement technique. At each iteration of the algorithm certain search directions or moves are forbidden (tabu) to avoid cycling and getting trapped at a local optimum. It has been successfully applied to a variety of combinatorial problems, e.g., traveling salesman [32], quadratic assignment [44], vehicle routing [15], graph coloring [19], bin packing [14], scheduling [2] and in functional synthesis of digital systems [1]. In fact, tabu search has given better solutions than what was known before, for some test problems (generally difficult to solve) for quadratic assignment problem [40] and bin packing [14].

### E. Basic Principles of Tabu Search

The TS algorithm starts with an initial feasible solution and moves successively to the *best* neighboring solution not visited before, to avoid being trapped in a local optima. The best neighboring solution need not always be better than the current one. Since, it is difficult to keep track of all previous solutions a common procedure is to keep track of all the modifications or moves carried out on a solution. Reversal of such moves is forbidden and are recorded on the tabu list (also called the short term memory). A move remains tabu only for a certain number of iterations as indicated by the length of tabu list. In principle, restricting new moves to only nontabu moves reduces the the risk of cycling, but can also result in rejecting some worthwhile moves (and hence solutions not visited yet). To prevent this, an aspiration level function is used to override the tabu status of a move, if the move is considered *good enough* by the criterion implicit in the function. Thus, an *admissible move* is either a nontabu move, or a tabu move satisfying the aspiration criterion. Intensification (extensive search in a promising region of the solution space) and diversification (investigating other regions of the solution space) of the search process are typically achieved by incorporating intermediate term and long term memory functions in the algorithm. Finally, the algorithm is terminated when a solution found is close enough to a good lower bound to the given problem (if available) or upon reaching a selected cutoff point. Since a good lower bound is not readily available for the problem under consideration, we terminate the algorithm when the best solution has not been updated for consecutive *max_iter* iterations, generally chosen as a function of problem size. In this study, *max_iter* is set at $3|I|$ based on trial runs.

### F. Tabu Restrictions and Diversification

Tabu restrictions are imposed to exclude moves which will bring the search process back to a solution visited before. A tabu restriction forbids an operation $i$ being reassigned to machine type $j$ to which it was assigned earlier until the short term memory has expired. This restriction is imposed by creating a $|I| \times |J|$ matrix called *tabu_time*, where the $(ij)$th element of the matrix contains the iteration number at which operation $i$ was moved from machine type $j$. Tabu restriction

is thus enforced if

$$tabu\_time_{ij} + tabu\_size \geq current\_iteration\_no$$

where *tabu_size* is the size of the tabu list. The size of the tabu list is critical for the success of the algorithm. Some applications have found the best value to lie between 5 and 12, clustered around 7, while others have found this value to depend on problem size and structure. A small value of *tabu_size* may permit cycling, whereas a large one may not help the search process, but will increase the computational time. Recent applications have suggested a dynamic short term memory to be more robust [14], [44]. Our procedure uses dynamic short term memory size in a random fashion to integrate the intensification and diversification strategies. The *tabu_size* is computed as follows:

$$tabu\_size = uniform(tabu\_min, tabu\_max)$$

where *tabu_min* and *tabu_max* are the minimum and maximum size of the short term memory, and *uniform* is a function that returns a random integer distributed uniformly between the limits. In addition, *tabu_size* is changed randomly during the search process after every two *tabu_max* iterations. The tabu list parameters *tabu_min* was set at four and *tabu_max* at 14.

As mentioned before, the tabu restrictions, if applied as described above may result in rejecting some good moves. An aspiration level function is used to provide an added flexibility in overruling the tabu restriction of a move. The tabu restriction of a move is overridden, if the aspiration level is attained. A simple aspiration level function often used is to override the tabu status of a move, if it results in an objective value strictly better than the best value obtained thus far. We have adopted an aspiration level function that overrides the tabu status of a move, if it results in a better objective value than the best value obtained when the move was made tabu. This condition does not require that the move result in a new best solution.

Diversification of the search process is important in finding good solutions to large problems. To achieve this, long term memory function is used to force the search process to visit regions not yet explored. It is normally activated after some initial period of search, in our case after the first $|I|$ iterations. This memory function is problem dependent and can be implemented in a variety of ways. In this application we have implemented the penalized frequency method. It uses a matrix *freq* whose $(ij)$th element indicates the number of times an operation $i$ has been reassigned from machine type $j$. This method is designed to create a bias against frequently executed moves. Diversification strategy is activated only when there is no admissible improving move, i.e., local region has been sufficiently explored. An adjusted processing time $t'_{ij}$ is computed for each $(ij)$ by appending a penalty to its processing time $t_{ij}$, i.e.,

$$t'_{ij} = t_{ij} + \gamma freq_{ij},$$

where $\gamma$ is a constant penalty parameter set at two (based on the initial trial runs). The best nonimproving move is chosen based on $t'_{ij}$ instead of $t_{ij}$.

### G. Move Generation

The efficient selection of the best move at each iteration is critical for the success of the algorithm and this depends on the solution space and move mechanism used. Let $\Omega$ be the feasible solution space for problem $\mathbf{P}$, then for a given solution $\omega \in \Omega$, its neighborhood $\Omega_\omega$ is defined as the set of all feasible solutions generated from $\omega$ by reassigning any operation $i \in I$, from its current assignment to machine type $j$ to another machine type $j'$. This gives a neighborhood size of $O(|I||J-1|)$, however, as we will show later, we generate a relatively small set of these moves that are likely to be superior.

In searching for the best move from a solution $\omega \in \Omega$ to an admissible solution in $\Omega_\omega$, we first try to see if an improving move (one that reduces the current makespan) is possible. This is achieved by reassigning one of the operations currently assigned to the machine type having the largest total processing time to another machine type such that the makespan reduces, provided that machine type has enough empty tool slots in its tool magazine to carry out the operation. This procedure is similar to the reassignment procedure described before, except we consider only the admissible moves, i.e., nontabu moves or tabu moves that satisfy the aspiration level criterion. In case an improving move is not possible we move to the nonimproving phase of the search process and reassign an admissible operation from its current assignment to another machine type that causes the makespan to increase by the least amount. It is possible that such a move may not result in increasing the current makespan. We also use an adjusted processing time $t'_{ij}$ (which incorporates a diversity measure using a frequency-based long-term memory function), instead of $t_{ij}$ in selecting a nonimproving move. Because of the inherent flexibility of an FMS in terms of operation assignment (due to machine capability), a feasible solution to the loading problem can be found most of the time using the following modified earliest completion time heuristic [22]

Step 0: Initialize. Let $T_j = 0 \quad \forall j \in J$, and $J_i^F = J_i \quad \forall i \in I$.

Step 1: If $I = \emptyset$, stop (all operations have been assigned).
Else let $J_i^C = \{j \in J_i^F : a_{ij} \leq b_j\} \quad \forall i \in I$.
If $J_i^C = \emptyset$ for any $i \in I$, then stop (no feasible solution found).

Step 2(a): Find $j(i) \in J_i^C$, such that $T_{j(i)} + t_{ij(i)} = \min_{j \in J_i^C}\{T_j + t_{ij}\} \quad \forall i \in I$.

Step 2(b): Find $i_o \in I$, such that $T_{j(i_o)} + t_{i_o j(i_o)} = \min_{i \in I}\{T_{j(i)} + t_{ij(i)}\}$.

Step 3: Assign operation $i_o$ to machine type $j(i_o)$ and update
$T_{j(i_o)} = T_{j(i_o)} + t_{i_o j(i_o)}$;
$b_{j(i_o)} = b_{j(i_o)} - a_{i_o j(i_o)}$.

Step 4: $I = I \backslash \{i_o\}$, and go to step 1.

However, in case the above heuristic yields an infeasible solution, one can easily solve $\{\max \Sigma_{i \in I} \Sigma_{j \in J_i} x_{ij} : \Sigma_{j \in J_i} x_{ij} \geq 1 \quad \forall i \in I$, (4) and (5)$\}$, using Lagrangian relaxation to get $J_i^F$ and then proceed with the above algorithm.

TABLE I
COMPARISON OF ALL THREE HEURISTICS

| Problem No | Problem Size $\|J\|$ | $\|I\|$ | $LB$ | Makespan(CPU time in sec) Heuristic 1 | Heuristic 2 | Heuristic 3 |
|---|---|---|---|---|---|---|
| 1 | 5 | 20 | 37 | 46 ( 0.25) | 43 ( 0.08) | 43 ( 0.11) |
| 2 | | | 39 | 43 ( 0.21) | 43 ( 0.04) | 43 ( 0.10) |
| 3 | | | 55 | 70 ( 0.26) | 64 ( 0.05) | 63 ( 0.11) |
| 4 | | 50 | 112 | 127 ( 0.77) | 131 ( 0.41) | 124 ( 0.70) |
| 5 | | | 110 | 120 ( 0.69) | 121 ( 0.47) | 119 ( 0.68) |
| 6 | | | 122 | 133 ( 0.62) | 133 ( 0.47) | 131 ( 0.81) |
| 7 | 10 | 50 | 58 | 72 ( 1.28) | 70 ( 0.89) | 67 ( 1.63) |
| 8 | | | 56 | 66 ( 1.64) | 64 ( 1.38) | 63 ( 1.28) |
| 9 | | | 54 | 63 ( 1.60) | 62 ( 1.46) | 61 ( 1.76) |
| 10 | | 75 | 78 | 86 ( 2.07) | 84 ( 1.93) | 84 ( 3.26) |
| 11 | | | 80 | 93 ( 2.09) | 89 ( 2.41) | 88 ( 3.11) |
| 12 | | | 86 | 100 ( 2.09) | 98 ( 2.26) | 97 ( 3.40) |
| 13 | | 100 | 113 | 126 ( 3.47) | 123 ( 6.09) | 122 ( 7.06) |
| 14 | | | 107 | 123 ( 3.22) | 120 ( 5.75) | 120 ( 6.52) |
| 15 | | | 107 | 118 ( 3.31) | 119 ( 6.06) | 117 ( 6.80) |
| 16 | 15 | 75 | 49 | 60 ( 5.31) | 59 ( 5.02) | 58 ( 5.83) |
| 17 | | | 54 | 69 ( 3.46) | 65 ( 5.51) | 64 ( 5.90) |
| 18 | | | 51 | 65 ( 4.10) | 61 ( 4.49) | 58 ( 5.79) |
| 19 | | 100 | 72 | 87 ( 5.59) | 83 ( 7.13) | 82 (11.95) |
| 20 | | | 72 | 88 ( 5.71) | 84 (10.25) | 84 (10.47) |
| 21 | | | 74 | 85 ( 6.37) | 83 ( 7.24) | 83 (15.53) |
| 22 | | 125 | 85 | 95 ( 9.46) | 93 (15.86) | 91 (30.17) |
| 23 | | | 88 | 100 (12.30) | 98 (11.52) | 96 (17.34) |
| 24 | | | 82 | 90 ( 7.83) | 89 (15.23) | 89 (21.75) |
| 25 | 20 | 100 | 51 | 60 (12.06) | 59 ( 9.84) | 58 (24.40) |
| 26 | | | 53 | 64 (10.62) | 63 (14.05) | 62 (21.55) |
| 27 | | | 49 | 60 ( 9.39) | 58 (16.11) | 59 (18.68) |
| 28 | | 125 | 69 | 84 (15.21) | 80 (26.67) | 79 (31.89) |
| 29 | | | 68 | 78 (14.36) | 79 (24.96) | 77 (53.60) |
| 30 | | | 67 | 81 (13.84) | 77 (23.43) | 76 (27.81) |
| 31 | | 150 | 81 | 89 (19.36) | 88 (32.87) | 87 (60.78) |
| 32 | | | 76 | 92 (21.53) | 86 (30.76) | 86 (38.76) |
| 33 | | | 80 | 92 (15.52) | 89 (32.74) | 89 (45.58) |
| 34 | 25 | 100 | 45 | 54 (12.79) | 54 (24.42) | 53 (30.32) |
| 35 | | | 42 | 55 (16.04) | 50 (19.88) | 51 (21.08) |
| 36 | | | 47 | 57 (20.82) | 56 (19.82) | 55 (23.08) |
| 37 | | 125 | 55 | 71 (17.63) | 67 (29.75) | 67 (33.06) |
| 38 | | | 54 | 64 (22.73) | 63 (31.71) | 62 (55.28) |
| 39 | | | 56 | 72 (19.80) | 66 (29.72) | 66 (69.11) |
| 40 | | 150 | 63 | 76 (31.38) | 72 (56.38) | 70 (65.59) |
| 41 | | | 69 | 80 (29.60) | 80 (48.90) | 79 (64.21) |
| 42 | | | 67 | 77 (43.88) | 77 (44.98) | 76 (71.98) |

**$|J|$: Number of machine types**
**$|I|$: Number of operations**
**$LB$: Lower bound on $v(P)$, from Heuristic 1**

### *H. Heuristic 3*

This heuristic is similar to Heuristic 2, however, it uses a hash function to control the tabu restrictions and diversification of the search process. Woodruff and Zemel [46] have proposed a novel approach of using the hash function within the TS framework to avoid cycling. Hashing is an important class of search methods in computer science, and is conceptually simple to implement and efficient for certain kinds of problems. A detailed description of the different kinds of hash functions and their use within the TS procedure is given in their paper.

Using the hashing scheme, each solution $\omega \in \Omega$ is mapped onto the hash list of size $m$, by computing some arithmetic function $h$ (also called the hash function), of $\omega$. In other words, $h(\omega)$ maps the solution space $\Omega$, onto the integers one through $m$. Ideally one would wish to design and use a hash function which transforms each solution into an unique integer, but this is difficult to achieve due to the combinatorial nature of our problem. Two of the most desirable properties of a hash function $h$ are its efficient computation and that it minimize the number of collisions. A collision occurs when two or more different solutions are mapped onto the same integer. Since the solution space $\Omega$, is usually much larger than the hash list, many different solutions may be mapped onto the same integer. A collision resolution technique is then required, to resolve these collisions. Before hashing the solutions, we carry out a preconditioning process, where a given solution $\omega \in \Omega$, is assigned a $solution\_key_\omega$, which can then be easily manipulated by the hashing function. The solution key for $\omega \in \Omega$ is defined as:

$$solution\_key_\omega = \sum_{i \in I} \sum_{j \in J_i} z_{ij} x_{ij}$$

where $z_{ij} = uniform(1, LARGE\_INTEGER)$ which re-

TABLE II
AVERAGE $LBPG$ AND CPU TIME FOR ALL THREE HEURISTICS

| Problem Size | | Average $LBPG$ | | | Average CPU time (sec) | | |
|---|---|---|---|---|---|---|---|
| $\|J\|$ | $\|I\|$ | Heuristic 1 | Heuristic 2 | Heuristic 3 | Heuristic 1 | Heuristic 2 | Heuristic 3 |
| 5 | 20 | 20.62 | 14.29 | 13.67 | 0.24 | 0.06 | 0.11 |
| | 50 | 10.50 | 11.99 | 8.76 | 0.69 | 0.45 | 0.73 |
| 10 | 50 | 19.55 | 16.60 | 13.66 | 1.51 | 1.24 | 1.56 |
| | 75 | 14.26 | 10.97 | 10.16 | 2.08 | 2.20 | 3.26 |
| | 100 | 12.25 | 10.74 | 9.82 | 3.33 | 5.97 | 6.79 |
| 15 | 75 | 25.89 | 20.13 | 16.87 | 4.29 | 5.01 | 5.84 |
| | 100 | 19.31 | 14.70 | 14.23 | 5.89 | 8.21 | 12.65 |
| | 125 | 11.72 | 9.77 | 8.23 | 9.86 | 14.20 | 23.09 |
| 20 | 100 | 20.28 | 17.64 | 17.04 | 10.69 | 13.33 | 21.54 |
| | 125 | 19.11 | 15.68 | 13.72 | 14.47 | 25.02 | 37.77 |
| | 150 | 15.31 | 11.02 | 10.60 | 18.80 | 32.12 | 48.37 |
| 25 | 100 | 24.08 | 19.40 | 18.74 | 16.55 | 21.37 | 24.83 |
| | 125 | 25.39 | 18.78 | 18.16 | 20.05 | 30.39 | 52.48 |
| | 150 | 17.17 | 15.05 | 13.01 | 34.95 | 50.09 | 67.26 |

$|J|$: Number of machine types
$|I|$: Number of operations

turns a random integer distributed uniformly between 1 and *LARGE_INTEGER*. It is important to note that each solution may not be assigned a unique solution key (although as *LARGE_INTEGER* increases the probability of different solutions having the same solution key decreases). We have set *LARGE_INTEGER* to 99 999, based on the system's (IBM 3090) limitations. Utilizing the same move generation scheme as in Heuristic 2, the solution key for a solution $\omega' \in \Omega_\omega$, generated from $\omega$ by the reassignment of an operation $i$ from machine type $j$ to $j'$ can be easily computed as

$$solution\_key_{\omega'} = solution\_key_{\omega} + z_{ij'} - z_{ij}$$

If the distribution of solution keys is known, a hash function can be developed which makes use of this distribution, however, when this information is not available, the hash function can be computed using methods similar to those of generating pseudorandom numbers. One of the most widely used and simple hashing functions is the division method defined as

$$h(\omega) = solution\_key_{\omega} \bmod m + 1$$

for some integer divisor $m$. The operator mod denotes the modulo arithmetic system. This function gives a hash list size of $m$. The best choice for $m$ is a large prime number which does not result in a biased use of the hash list. In practice, it has been found that it is sufficient to choose $m$ with no prime divisors less than 20 [20]. Based on the system's (IBM 3090) limitations, we have set $m$ to 91813, which meets the above criteria. In summary, to effectively use the hashing function within the TS procedure, we need to minimize collision of two different solutions having the same solution key and two different solution keys getting mapped onto the same integer on the hash list.

The tabu restrictions and diversification of the search process can be easily implemented by using the hash list as described next. A move from solution $\omega$ to $\omega' \in \Omega_\omega$ is admissible, if it is not present on the hash list, or if present on the hash list meets the aspiration level criterion. An aspiration level is defined for each *solution_key* encountered during the search process, in a manner similar to the one described for Heuristic 2. An admissible move from $\omega$ to $\omega' \in \Omega_\omega$ is either not present on the hash list, or if present on the hash list meets the aspiration level criterion. Thus, aspiration level is used to resolve the collision on the hash list. The selection of the best move is essentially the same as that for Heuristic 2, except the constant penalty parameter $\gamma$ is set to zero, during the non improving phase of the search process. The initial solution for this heuristic is also obtained using the same procedure as that for Heuristic 2.

## IV. COMPUTATIONAL RESULTS

All algorithms were written in FORTRAN and run under VM/CMS on an IBM 3090 computer. A set of computational experiments were carried out to test the performance of each heuristic. The problem data used in these experiments were generated randomly, but systematically to capture the range of different problems encountered in practice [27], [38]. For a given problem size (number of operation and machine types), the processing time $t_{ij}$ for each operation $i$ on machine type $j$ was generated from *uniform* (6, 30). Eighty percent of the tool types require one tool slot, whereas the remaining twenty percent require three tool slots. The number of tool slots $a_{ij}$, required for an operation $i$ on machine type $j$ is computed as $\Sigma_{l=1}^{L_{ij}}\ g_l$, where $g_l$ is the number of tool slots required for tool $l$, and $L_{ij}$ is the number of tool types required to carry out the operation, and is generated from *uniform* (10, 30). In addition each operation can have up to four optional machine types to process it. Finally, the tightness of tool magazine capacity constraint was set at 0.60, i.e., $b_j = 0.60\ \Sigma_{i \in I_j}\ a_{ij}$. The stopping criteria for all three heuristics have been described before and are based on the initial trial runs. Similar stopping criteria have been used before for the solution of other combinatorial problem using these techniques [12], [18]. Essentially, the algorithms are terminated when there is very little or no possibility for the best solution found to be updated.

Table I summarizes the results in terms of makespan and the computational time (cpu seconds) obtained from all three heuristics. The problem size ($|J|$ and $|I|$) considered here range from small to reasonably large for practical applications.

Three replicates are reported for each case for a total of 42 problems. We feel that this set of test problems is sufficient to illustrate the performance and complexity of all three heuristics. The lower bound $(LB)$ provided by the Lagrangian relaxation can be used as a comparison base. From these results, one can conclude that all three heuristics have provided good quality solutions in a reasonable amount of cpu time. However, TS based heuristics have provided better solutions than the Lagrangian based heuristic. This can further be demonstrated by computing a lower bound percentage gap $(LBPG)$ for all three solution approaches. A $LBPG_{(\cdot)}$ for heuristic $(\cdot)$ can be computed as

$$LBPG_{(\cdot)} = (v(P)_{(\cdot)} \dot{-} LB)/LB * 100,$$

where $v(p)_{(\cdot)}$ is the makespan from heuristic $(\cdot)$.

Table II gives the average value of $LBPG$ and cpu time over three replicates for each combination of $|J|$ and $|I|$. Heuristic 2 and 3 have produced significantly better quality solution than Heuristic 1, with Heuristic 3 (TS with hashing) consistently outperforming the other two. The computational effort required to solve these problems using the three heuristics is not much, although as the problem size increases, Heuristic 3 requires considerably more effort than the other two heuristics. This can be inferred by comparing the average cpu time for all three heuristics in Table II. One of the reasons why Heuristic 3 requires more time is the computation of mod function (expensive to evaluate) several times during the search process. It is important to remember that Heuristic 3 is also easier to implement than the other two heuristics, for example Heuristic 2 (TS) requires fine tuning of some of its parameter values (like *tabu_size*, $\gamma$, etc), whereas the Heuristic 3 uses hashing to control the tabu restriction and diversification and frees the user of the need to specify these parameter values. Performance of Heuristic 1 can be improved by embedding the Lagrangian relaxation in a branch and bound procedure. Thus, our computational results indicate that both the Lagrangian and the TS-based heuristics have been able to provide good results in a reasonable amount of time.

## V. Conclusion

In this paper, we have developed effective heuristics for solving the loading problem in FMS's and illustrated their performance on a set of small to reasonably large practical sized problems. Our computational experience suggests that TS-based heuristics outperform the Lagrangian-based heuristic and in particular TS with hashing was very effective in controlling the tabu restrictions and diversification of the search process. In addition, we feel that TS with hashing is easier to implement than the other heuristics. For future study, some of the solution techniques developed here can be applied to more complex loading models as well as to other planning subproblems. Furthermore, an interesting area of research would be to combine batching and loading with scheduling for specific FMS described in [31], and develop efficient solution techniques to solve them.

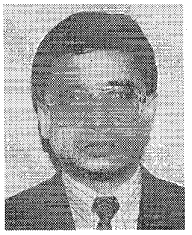
## Acknowledgment

The authors would like to thank the Associate Editor and the referees for their careful review and constructive comments on earlier versions of this paper.

## References

[1] S. Amellal and B. Kaminska, "Functional synthesis of digital systems with TASS," *IEEE Trans. Computer-Aided Design*, vol. 13, pp. 537–552, 1994.
[2] J. W. Barnes and M. Laguna, "A tabu search experience in production scheduling," *Ann. Oper. Res.*, vol. 41, pp. 141–155, 1993.
[3] M. S. Bazaara and J. J. Goode, "A survey of various tactics for generating Lagrangian multipliers in the context of Lagrangian duality," *Eur. J. Oper. Res.*, vol. 3, pp. 322–338, 1979.
[4] M. S. Bazaara and C. M. Shetty, *Nonlinear Programming: Theory and Algorithms*. New York: Wiley, 1979.
[5] M. Berrada and K. E. Stecke, "A branch and bound approach for machine load balancing in flexible manufacturing systems," *Manage. Sci.*, vol. 32, pp. 1316–1335, 1986.
[6] J. Buzacott, "Modeling manufacturing systems," *Robot. Comput.-Integr. Manuf.*, vol. 2, pp. 25–32, 1985.
[7] A. K. Chakravarty and A. Shtub, "Selecting parts and loading flexible manufacturing systems," *Proc. First ORSA/TIMS Special Interest Conf. on Flexible Manufacturing Systems*, pp. 284–289, 1984.
[8] L. F. Escudero, "Production planning modeling of FMS," in *Modeling the Innovation: Communication, Automation and Information Systems*. Amsterdam: North-Holland, 1990, pp. 231–238.
[9] M. L. Fisher, "The Lagrangian relaxation method for solving integer programming problems," *Manage. Sci.*, vol. 27, pp. 1–18, 1981.
[10] S. W. Flanders and W.J. Davis, "Scheduling a flexible manufacturing system with tooling constraints: An actual case study," *Interfaces*, vol. 25, pp. 42–54, 1995.
[11] A. M. Geoffrion, "Lagrangean Relaxation for Integer Programming," *Math. Prog. Study*, vol. 2, pp. 82–114, 1974.
[12] F. Glover, "Tabu search, Part I," *ORSA J. Comput.*, vol. 1, pp. 190–206, 1989.
[13] ——, "Tabu search, Part II," *ORSA J. Comput.*, vol. 2, pp. 4–32, 1990.
[14] F. Glover and R. Hubscher, "Bin packing with tabu search," Tech. Rep., Graduate School of Business Administration, Univ. of Colorado at Boulder, 1991.
[15] M. Gendreau, A. Hertz, and G. Laporte, "A tabu search heuristic for vehicle routing problem," *Manage. Sci.*, vol. 40, pp. 1276–1290, 1994.
[16] A. M. A. Hariri and C. N. Potts, "Heuristics for scheduling unrelated parallel machines," *Comput. Oper. Res.*, vol. 18, pp. 323–331, 1991.
[17] M. Held and R. M. Karp, "The traveling salesman problem and minimum spanning trees," *Oper. Res.*, vol. 18, pp. 1138–1162, 1970.
[18] M. Held, P. Wolfe, and H. O. Crowder, "Validation of subgradient optimization," *Math. Prog.*, vol. 6, pp. 62–88, 1974.
[19] A. Hertz and D. de Werra, "Using tabu search techniques for graph coloring," *Comput.*, vol. 39, pp. 345–351, 1987.
[20] E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms*. Rockville, MD: Computer Science, 1978.
[21] S. Hwan and A. Shogan, "Modeling and solving an FMS part selection problem," *Int. J. Prod. Res.*, vol. 27, pp. 1349–1366, 1989.
[22] O. H. Ibarra and C. E. Kim, "Heuristic algorithms for scheduling independent tasks on nonidentical processors," *J. Assoc. Comput. Mach.*, vol. 24, pp. 280–289, 1977.
[23] R. Jaikumar and L. N. Van Wassenhove, "A production planning framework for flexible manufacturing systems," *J. Manuf. Oper. Manage.*, vol. 2, pp. 52–79, 1989.
[24] Y. D. Kim and C. Yao, "Heuristic approaches for loading problems in flexible manufacturing systems," *IIE Trans.*, vol. 25, pp. 26–39, 1993.
[25] ——, "A due date based approach to part type selection in flexible manufacturing systems," *Int. J. Prod. Res.*, vol. 32, pp. 1027–1044, 1994.
[26] P. Kouvelis, "An optimal tool selection procedure for the initial design phase of a flexible manufacturing system," *Eur. J. Oper. Res.*, vol. 55, pp. 201–210, 1991.
[27] P. Kouvelis and H. L. Lee, "Block angular structures and the loading problem in flexible manufacturing systems," *Oper. Res.*, vol. 39, pp. 666–676, 1991.
[28] A. Kusiak, "Loading models in flexible manufacturing systems," *Recent Developments in Flexible Manufacturing Systems and Allied Areas*. Amsterdam: North-Holland, 1984, pp. 119–132.
[29] R. C. Leachman and T. F. Carmon "On capacity modeling for production planning with alternative machine types," *IIE Trans.*, vol. 24, pp. 62–72, 1992.

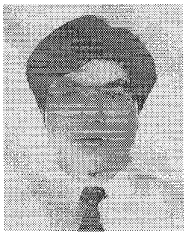
[30] M. Liang and S. P. Dutta, "Solving a combined part-selection, machine-loading, and tool-configuration problem in flexible manufacturing systems," *Prod. Oper. Manage.*, vol. 2, pp. 97–113, 1993.
[31] B. L. MacCarthy and J. Liu, "A New Classification Scheme for Flexible Manufacturing Systems," *Int. J. Prod. Res.*, vol. 31, pp. 299–309, 1993.
[32] M. Malek, M. Guruswamy, H. Owens, and M. Pandya, "Serial and parallel search techniques for the traveling salesman problem: theory and applications," *Ann. Oper. Rese.*, vol. 21, pp. 26–51, 1989.
[33] E. Mansfield, "The diffusion of flexible manufacturing systems in Japan, Europe and the United States," *Manage. Sci.*, vol. 39, pp. 149–159, 1993.
[34] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*. Chichester, England: Wiley, 1990.
[35] S. Rajagopalan, "Formulation and heuristic solutions for parts grouping and tool loading in flexible manufacturing systems," *Proc. S2nd ORSA/TIMS Conf. on FMS: Operations Research Models and Applications*, 1986, pp. 311–320.
[36] R. Ram, S. C. Sarin, and C. S. Chen, "A model and a solution approach for the machine loading and tool allocation problem in a flexible manufacturing system," *Int. J. Prod. Res.*, vol. 28, pp. 637–645, 1990.
[37] S. C. Sarin and C. S. Chen, "The machine loading and tool allocation problem in an FMS," *Int. J. Prod. Res.*, vol. 25, pp. 1081–1094, 1987.
[38] K. Shanker and Y. J. Tzen, "A loading and dispatching problem in a random flexible manufacturing system," *Int. J. Prod. Res.*, vol. 23, pp. 579–595, 1985.
[39] K. Shanker and A. Srinivasulu, "Some solution methodologies for loading problems in flexible manufacturing system," *Int. J. Prod. Res.*, vol. 27, pp. 1019–1034, 1989.
[40] J. Skorin-Kapov, "Tabu search applied to the quadratic assignment problem," *ORSA J. Comput.*, vol. 2, pp. 33–45, 1990.
[41] K. E. Stecke, "Formulation and solution of nonlinear integer production planning problems for flexible manufacturing systems," *Manage. Sci.*, vol. 29, pp. 273–288, 1983.
[42] K. E. Stecke, "A hierarchical approach to solving machine grouping and loading problems of flexible manufacturing systems," *Eur. J. Oper. Res.*, vol. 24, pp. 369–378, 1986.
[43] K. E. Stecke and F. B. Talbot, "Heuristics for loading flexible manufacturing systems," in *Recent Developments in FMS, Robotics, CAD/CAM, CIM*. Amsterdam: North-Holland, 1985, pp. 73–85.
[44] E. Taillard, "Robust taboo search for the quadratic assignment problem," *Parallel Comput.*, vol. 17, pp. 443–455, 1991.
[45] A. J. Van Looveren, L. F. Gelders, and L. N. Van Wassenhove, "A review of FMS planning models," in *Modeling and Design of Flexible Manufacturing Systems*. Amsterdam: Elsevier, 1986, pp. 3–31.
[46] D. L. Woodruff and E. Zemel, "Hashing vectors for tabu search," *Ann. Oper. Res.*, vol. 41, pp. 123–140, 1993.



**Bharatendu Srivastava** received the B. Tech degree in chemical engineering from the Indian Institute of Technology, and the M.B.A and Ph.D. degrees in decision sciences from Washington State University, Pullman.

He is currently an Assistant Professor of Operations Management at Marquette University, Milwaukee, WI. His papers have appeared or will appear in *Annals of Operations Research, Discrete Applied Mathematics, European Journal of Operational Research, International Journal of Computer Integrated Manufacturing,* and *International Journal of Production Economics.* He serves on the editorial review board of *International Journal of Operations and Quantitative Management.* His research interests are in production planning and scheduling for FMS, group technology, cellular manufacturing and integrated product/process design.

Dr. Srivastava is a member of the Decision Sciences Institute, INFORMS, Production and Operations Management Society, and APICS.



**Wun-Hwa Chen** received the B.S. degree from National Chiao-Tung University, Taiwan, in 1980, and the M.B.A. and Ph.D. degrees in management science from the State University of New York at Buffalo, in 1983 and 1989, respectively.

From 1989 to 1994, he was an Assistant Professor, Department of Management and Systems, Washington State University, Pullman. He is currently an Associate Professor, Department of Business Administration at the National Taiwan University. His research and teaching interests are in the areas of production planning, heuristic algorithms, combinatorial optimization and logistics system. He held a summer appointment at the University of New South Wales, Sydney, Australia in 1995.