Combining Representations from Manufacturing, Machine Planning, and Manufacturing Resource Planning (MRP)

Billy Harris wharris@cse.uta.edu
Diane Cook cook@cse.uta.edu
Department of Computer Science and Engineering
University of Texas at Arlington
Arlington, Texas 76019

Frank Lewis flewis@arri.uta.edu
Automation and Robotics Research Institute
7300 Jack Newel Blvd South
Ft. Worth, Texas 76118

Abstract

We integrate an ordinary planner into a manufacturing system by showing that the assembly trees used by manufacturers can be converted into domain operators and that the operator sequence formed by the planner can be converted into a set of matrices used by the manufacturing system. This allows manufacturers to continue to use their existing representations where possible. We form additional resource matrices based on the planner's output which an existing dispatching system uses to reserve machines and avoid deadlock. We also show how our planner and matrix notation can efficiently implement Manufacturing Resource Planning. In many cases, our MRP system can seamlessly integrate limited production capacity without manager intervention.

Introduction

Previous research has developed a real-time control system using matrices to describe the sequence of operations and resources needed to construct parts and subassemblies (Tacconi & Lewis 1997). We have added AI planning technology to this system and generated these matrices which were previously hand-generated. We have shown that manufacturing assembly trees can be encoded into HTN operators which allow both input and output from our planner to use manufacturing representations. Although the system originally focused on real-time control, we have shown that our matrix-based approach can also be used for inventory control and advance planning traditionally done by MRP systems.

We begin by giving a brief overview of the real-time controller. The controller uses four matrices: F_v and S_v describe ordering constraints between plan operations, and F_r and S_r describe resource constraints.

We show how assembly trees can be used to form plan operators and describe how the planner's output can be converted into the F_v and S_v matrices. We introduce generic resources and use a resource assignment matrix to form the F_r and S_r matrices from the F_v and S_v matrices. One key advantage that machine planners can offer is the ability to easily handle alternate courses of action—called job-shop scheduling by manufacturers. We describe how our planner can form multiple F_v and S_v matrices and how these can be combined into a single set of matrices.

The matrices formed by our planners can also be used for the long-term planning horizons used by Manufacturing Resource Planning. We describe the MRP problem and show how our matrix notation enhances the MRP algorithm by allowing limited production capacity

Inputs to Control System

Researchers studying issues in intelligent control at the Automation and Robotics Research Institute (ARRI) have developed a multi-level real-time discrete event control system (Tacconi & Lewis 1997). The control system needs as input four matrices describing the conditions under which the system can legally transition between job steps. Figure 1 shows a sample set of matrices.

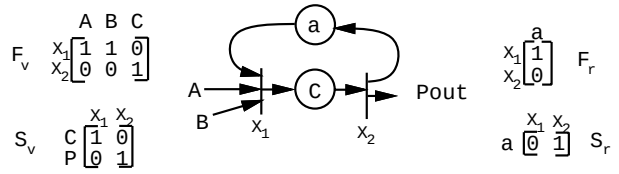

Figure 1: Sample Fv and Sv Matrices

The F_v matrix maps actions to transitions; a 1 in location *i, j* means that transition X_i cannot fire until action A_j completes. The S_v matrix maps transitions into actions; a 1 in location *i, j* of this matrix means that when transition X_j fires, action A_i is started. Two or more 1s in a single row of F_v signal assembly operations. Two or more 1s in a single column of F_v signal the start of a job-shop choice. Two or more 1s in the same row of S_v indicate the end of a job shop choice. Our controller does not allow the same column of S_v to have more than a single 1.

F_r and S_r describe the resource requirements; a 1 in position *i, j* of the F_r matrix means that resource R_j must be secured before transition X_i can fire. A 1 in position *i, j* in the S_r matrix means that when transition X_j fires, resource R_i is released. If a column *j* of F_r contains more

than one 1, resource *j* is being shared by more than one job.

Although the controller implementation uses matrix operations, the matrices can be viewed as describing a Petri-Net (See Figure 1). Using this view, the F_v and S_v matrices describe places corresponding to tasks; a token in a task place corresponds to a completed operation. Places in F_r and S_r correspond to resources; tokens in resource places correspond to the availability of that resource to manufacturing operations.

We emphasize that the control system, including the matrix representation, was developed without consideration of machine planning technology. The matrices, previously hand-generated, capture information genuinely needed by the control system in the form most helpful to the control system. By converting our planner's output to this representation, we insure the planner is both genuinely improving the system and also easily integrated with the system.

Converting Assembly Trees to HTN Operators

Manufacturers use assembly trees (Wolter, Chakrabarty, & Tsao 1992) or, equivalently, bill of materials (BOM) (Baker 1974) to specify a partial order of jobs required to complete a finished product. Assembly trees do not contain any information about the resources needed for the jobs; they contain only product-specific job sequencing information. We represent assembly trees graphically, with an edge for each manufacturing operation. Nodes in the graph represent parts or sub-assemblies; the part name changes as operations are performed. Figure 2 shows a sample assembly tree; by drilling part *A*, we create part *B*. Parts *B* and *C* can be assembled to form the single part *D*.

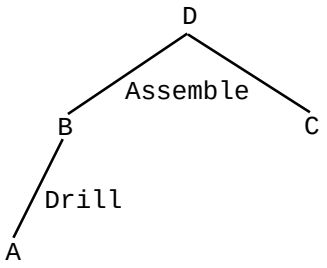

Figure 2: Sample Assembly Tree

In this section, we describe how to convert nodes in an assembly tree into HTN operators. We use the operator syntax from UM-Nonlin (Ghosh *et al* 1992), but the descriptions are easily converted into other operator description languages. Like other HTN operators, UM-Nonlin operators explitly identify the subgoal(s) achieved by each operator; in UM-Nonlin this is in the :todo field. The :expansion part shows subgoals and primitive actions needed to execute the operator, and the :effects part describes the result of the operator's execution. An :orderings field indicates the partial-order constraints on the subgoals and primitive actions listed in the :expansion field.

The label of an assembly tree node names the part produced. This becomes the operator's :todo and :effects. Each of the node's children becomes a subgoal The action needed to produce the part, listed on the arcs linking the part to its subassemblies, becomes a primitive (directly executable) action for our planner. We complete the operator by specifying that the primitive action accomplishes the operator's goal. Ordering constraints are simple; each subgoal (child node of the tree) must be complete before the primitive action (arc label) can begin. Figure 3 summarizes these correspondences; the resulting plan operators appear in the top portion of Figure 4.

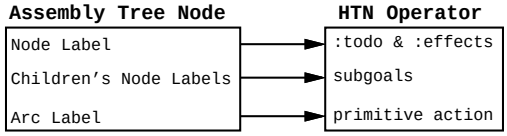

Figure 3: Converting Tree Nodes to Operators

```
(actschema Build-B
        :todo (Assembled B)
        :expansion ((step1 :goal (Assembled A))
                    (step2 :primitive (Drill A)))
        :effects ((step2 :assert (Assembled B)))
        :orderings ((step1 -> step2)))
(actschema Build-D
        :todo (Assembled D)
        :expansion ((step1 :goal (Assembled B))
                    (step2 :goal (Assembled C))
                    (step3 :primitive (Attach B C)))
        :effects ((step3 :assert (Assembled D)))
        :orderings ((step1 -> step3) (step2 -> step3))
(actschema Prepare-A
        :todo (Assembled A)
        :expansion ((step1 :primitive (PutOn A Pallet)
        :effects ((step1 :assert (Assembled A))))
```

Figure 4: Operators for Sample Assembly Tree

Leaf nodes for an assembly tree represent incoming parts. Our planner can use "stub" operators to indicate the point in the plan when the incoming parts are required. We do this by including HTN operators with no subgoals, as shown in the bottom of Figure 4 for the incoming part A.

Assembly trees show only a single method of contructing a part, but plan operators can represent many alternate means of constructing a part. Manufacturers refer to this as a job-shop scheduling problem; we detail this ability in a later section.

Converting Plan to Task Matrices

Figure 5 shows a sample plan which includes assembly operations and partial ordering of steps (operations F and G may be done in either order or simultaneously). The choice of orderings leads to a simple job shop choice. Our controller represents job-shop choices as an explicit choice between job sequences, so when we convert the plan to the controller's representation, we explicitly show the two choices: The controller can execute F and then G (corresponding to executing operations F1 and G1) or it can execute G and then F (corresponding to operations G2 and F2). In (Harris, Cook, & Lewis 200), we show how

this plan, after the partial orderings are expanded, can be converted into the F_v and S_v matrices shown in Figure 6.

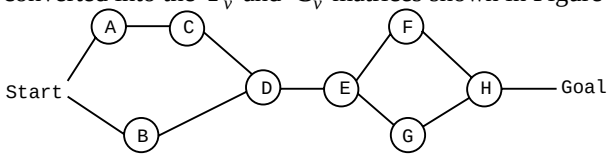

Figure 5: A Sample Plan

$F_V =$

	P_{inA}	P_{inB}	A	B	C	D	E	F1	G1	G2	F2	H
x_1	1	0	0	0	0	0	0	0	0	0	0	0
x_2	0	1	0	0	0	0	0	0	0	0	0	0
x_3	0	0	1	0	0	0	0	0	0	0	0	0
x_4	0	0	0	1	1	0	0	0	0	0	0	0
x_5	0	0	0	0	0	1	0	0	0	0	0	0
x_6	0	0	0	0	0	0	1	0	0	0	0	0
x_7	0	0	0	0	0	0	1	0	0	0	0	0
x_8	0	0	0	0	0	0	0	1	0	0	0	0
x_9	0	0	0	0	0	0	0	0	1	0	0	0
x_{10}	0	0	0	0	0	0	0	0	0	1	0	0
x_{11}	0	0	0	0	0	0	0	0	0	0	1	0
x_{12}	0	0	0	0	0	0	0	0	0	0	0	1

$S_V =$

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}
A	1	0	0	0	0	0	0	0	0	0	0	0
B	0	1	0	0	0	0	0	0	0	0	0	0
C	0	0	1	0	0	0	0	0	0	0	0	0
D	0	0	0	1	0	0	0	0	0	0	0	0
E	0	0	0	0	1	0	0	0	0	0	0	0
F1	0	0	0	0	0	1	0	0	0	0	0	0
G1	0	0	0	0	0	0	1	0	0	0	0	0
G2	0	0	0	0	0	0	0	1	0	0	0	0
F2	0	0	0	0	0	0	0	0	1	0	0	0
H	0	0	0	0	0	0	0	0	0	1	1	0
P_{out}	0	0	0	0	0	0	0	0	0	0	0	1

Figure 6: F_v and S_v Matrices

Initially, we assume that each operation has its own dedicated resource. That is, if a transition starts action A_i, it will also reserve a dedicated resource R_i, and if the completion of an action A_j causes a transition to fire, the transition will also release a resource R_j. This correspondence allows us to quickly form initial estimates for the resource matrices:

$\hat{F}_r = S_v^T$, with the product-out column(s) removed.

$\hat{S}_r = F_v^T$, with the product-in row(s) removed.

In most cases, this initial assumption is unrealistic. For example, if actions A, D, and E all involve drilling operations but we only have one drill, our drill must be shared by the three operations. In terms of dedicated resources, this sharing means that $\hat{a}$, d, and $\hat{e}$ all correspond to the same actual resource. We represent the sharing in a resource assignment matrix F_a. If F_a contains a 1 in position i,j, then our actual resource R_j is performing the duties of our idealized dedicated resource R_i. Shared resources are represented by columns of F_a containing two or more 1s.

Figure 7 shows a resource assignment matrix containing two shared resources; a single resource *ade* performs the functions of the generic resources $\hat{a}$, d, and $\hat{e}$ (and will be shared among actions *A*, *D*, and *E*) and a single resource f will perform the functions of generic resources $f1$ and $f2$ (and thus will be shared by actions *F*1 and *F*2).

$F_a =$

	ade	b	c	f	g1	g2	h
$\hat{a}$	1	0	0	0	0	0	0
$\hat{b}$	0	1	0	0	0	0	0
$\hat{c}$	0	0	1	0	0	0	0
d	1	0	0	0	0	0	0
$\hat{e}$	1	0	0	0	0	0	0
$f1$	0	0	0	1	0	0	0
$\hat{g1}$	0	0	0	0	1	0	0
$\hat{g2}$	0	0	0	0	0	1	0
$f2$	0	0	0	1	0	0	0
$\hat{h}$	0	0	0	0	0	0	1

Figure 7: F_a Assigns Resources

Once we have formed our resource assignment matrix, we can easily compute our final S_r and F_r matrices.

$$F_r = \hat{F}_r * F_a$$

$$S_r = F_a^T * \hat{S}_r$$

One slight complication can arise with self-loops. For example, consider resource *ade*. With dedicated resources, transition X_5 reserves $\hat{e}$ and releases d. Now, transition X_5 reserves resource *ade* and releases the same resource *ade*. This behavior is not correct; intuitively, it means that at the instant X_5 fires, two uses of resource *ade* are held. We eliminate this self-loop by finding *i,j* pairs such that $F_r[i,j] = S_r[j,i] = 1$ and changing both values to 0. This action corresponds to three matrix equations, in which "&" represents an element-by-element logical AND operation and "-" represents ordinary matrix subtraction:

$$T_s = F_r \,\&\, S_r^T$$

$$F_{r_{new}} = F_{r_{old}} - T_s$$

$$S_{r_{new}} = S_{r_{old}} - T_s^T$$

Figure 8 shows our final S_r and F_r matrices. To summarize, the plan itself forms S_v and F_v matrices once partial orderings are converted to an explicit choice of total orderings. The idea of generic resources allows us to form initial resource matrices; by using a resource assignment matrix F_a, we can easily convert the initial resource matrices into the final F_r and S_r matrices. Thus, our planner has successfully formed the four task matrices needed by the manufacturing control system.

$F_r =$

	ade	b	c	f	g1	g2	h
x_1	1	0	0	0	0	0	0
x_2	0	1	0	0	0	0	0
x_3	0	0	1	0	0	0	0
x_4	1	0	0	0	0	0	0
x_5	0	0	0	0	0	0	0
x_6	0	0	0	1	0	0	0
x_7	0	0	0	0	1	0	0
x_8	0	0	0	0	0	1	0
x_9	0	0	0	1	0	0	0
x_{10}	0	0	0	0	0	0	1
x_{11}	0	0	0	0	0	0	1
x_{12}	0	0	0	0	0	0	0

$S_r =$

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}
ade	0	0	1	0	0	1	1	0	0	0	0	0
b	0	0	0	1	0	0	0	0	0	0	0	0
c	0	0	0	1	0	0	0	0	0	0	0	0
f	0	0	0	0	0	0	0	1	0	0	1	0
g1	0	0	0	0	0	0	0	0	1	0	0	0
g2	0	0	0	0	0	0	0	0	0	1	0	0
h	0	0	0	0	0	0	0	0	0	0	0	1

Figure 8: F_r and S_r Matrices

Managing Multiple Plans

One advantage machine planners offer over conventional manufacturing representations is that they can easily represent alternate methods of constructing a part.

In (Gracanin, Srinivasan, & Valavanis 1994), Gravcanin uses a parameterized Petri-net to represent different means of assembling an *S4* part. We have shown the alternatives in assembly tree form in Figure 9.

Of course, machine planners can very easily generate alternate plans based on different operator sequences; in (Harris, Cook, & Lewis 2000), we have described a polynomial-time algorithm for combining different plans into a unified set of matrices. The algorithm begins by forming separate F_v and S_v pairs for each solution. It forms a mapping between places in the second plan and (possibly new) places in the first plan; based on this mapping, it determines which transitions from the second plan are novel and adds equivalent transitions to the first plan. Thus, the first set of F_v and S_v matrices will incorporate alternatives from both plans. Once the planner has formed these matrices, we can assign resources and form F_r and S_r as previously described.

Combining alternatives in this way allows our dispatcher to dynamically decide among alternate courses of action based on such dynamic factors as which machines are available, what order products arrive in, etc. In other words, the dispatcher can take advantage of conditions at execution time to decide among alternatives which are equally viable at planning time.

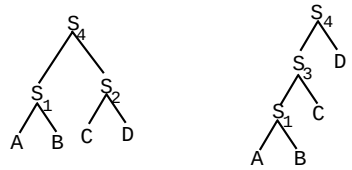

Figure 9: Alternate Assembly Trees

Manufacturing Resource Planning

Although the matrix notation was originally designed for a real-time control system, the S_v and F_v matrices can also represent the ordering constraints needed for Manufacturing Resource Planning (MRP). In this section, we show how our matrix notation allows for an efficient implementation of a Manufacturing Resource Planning system which accounts for limited capacity constraints for a large class of systems. In this paper, MRP refers to Manufacturing Resource Planning (MRP-II) and not the outdated Materials Requirement Planning (little–mrp).

Traditional MRP Implementation

This section describes an implementation of MRP for a single part. The algorithm uses several vectors as shown in Table 1. Each vector tracks the quantity of a particular part over time. Later, we will consider each vector to be one row of a matrix.

Table 1: Vectors and Scalars used by MRP

Name	Type	Description
R	vector	Gross Requirements; amount of product required at each time period.
λ	scalar	Lead time; the time need to produce (or order) a product.
I	vector	Inventory; records the amount of product on hand at each time step.
$I_{\min}$	scalar	Minimum inventory; the amount of safety stock to keep on hand.
S	vector	Scheduled Receipts; amount of product we have already planned for.
N	vector	Net requirements; needs in excess of inventory.
POR	vector	Planned Order Receipt;amount of product we plan to purchase/assemble.
κ	scalar	Capacity constraint; maximum number produced per time unit
es	vector	Orders in excess of capacity.
O	vector	Adjusted planned receipt
E	vector	Planned Order Release; amount of product we must begin assembling.

Inputs. The basic algorithm forms vectors for a single part and takes as input two vectors and three scalars. The R vector gives the gross requirements—a list of how many parts we need at each time unit. The lead time, λ, quantifies the time that will pass between issuing an order for a part and receiving the completed part. I_0, the initial inventory, gives the current stock-on-hand for the part. I_{min}, the minimum inventory, gives the amount of safety stock to keep on hand at all times. Finally, the S vector gives the scheduled receipts; it indicates how many parts are already in production and when they will be ready.

Forming POR. We use a streamlined implementation of MRP based on equations from (Bedworth & Bailey 1987) to form POR, the Planned Order Receipt vector. This process also forms *I*, the inventory vector. These vectors are constructed piecewise, starting from t_0 and working upwards:

$$N_t = R_t - S_t - I_t + I_{\min}$$
$$POR_t = \max(0, N_t)$$
$$I_{t+1} = \max(I_{\min}, -N_t)$$

The equations work by forming *N*, the Net Requirements vector, one element at a time. A negative element of *N* indicates that supply exceeds demand; the excess will be stored in inventory and appear in the *I* vector. Positive elements of *N* indicate that demand exceeds supply and thus additional parts must be purchased or built. These parts appear in the POR vector.

Capacity Contraints. The POR vector assumes that our factory has unlimited production ability. However, even at a long-term view, our factory is limited in the number of parts it can produce per time unit. We use κ to indicate the capacity constraint for the current part. We can seamlessly account for this limited capacity by starting from the highest time unit and working backwards toward t_0:

$$O_t = \max(POR_t + es_t, \kappa)$$
$$es_{t-1} = \max(0, POR_t + es_t - \kappa)$$

These equations form the vector *es* which indicates demand in excess of production capacity. A non-zero value for an element of es indicates that future demand will exceed production capacity and thus the factory must begin work at an earlier time period and store items in inventory. The equations form a value for es_{-1}; a non-zero value for es_{-1} means that the schedule is infeasible; even if the factory immediately begins operating at full capacity, it will be unable to satisfy its requirements. Assuming that the schedule is feasible, the O vector (Adjusted Order Receipt) indicates a schedule for when parts should finish being manufactured or arrive from suppliers.

Lead Time. Our manufacturing operations may take more than one time unit to complete. Likewise, some purchased products must be ordered well in advance. The lead time (λ) indicates how long this takes. We use matrices we call ∂-matrices to account for lead time. The ∂_1 matrix has 1s on its first sub-diagonal; when a vector is multiplied by ∂_1, the vector will be left-shifted by one unit. Similarly, ∂_2 has 1s on the second subdiagonal and when a vector is multiplied by ∂_2, it will be left-shifted by two units. These matrices make it easy to account for lead-times:

$$E = O * \delta_\lambda$$

The E vector (Planned Order Release) is the main result of MRP. It indicates when manufacturing must begin or when purchase orders must be placed for the part. In particular, if E has a non-zero value at t_0, then a purchase order or manufacturing authorization must be placed during the current time unit to avoid a slipped schedule.

Matrix MRP Implementation

We have developed an MRP implementation which works on the matrices developed by our planner. We present an extended description of our algorithm, with an example, in (Harris, Lewis, & Cook 2000b). Table 2 shows the matrices used by our algorithm; we maintain separate matrices for manufacturing operations and purchasing operations. In the table, *i* is the number of manufactured parts, *l* is the number of purchased parts, and *h* is determined by the planning horizon.

Our matrix algorithm (shown in Figure 10) accepts as input **R**, the requirements matrix. This matrix indicates the duetimes for all parts considered by our system. We repeatedly extract a row from **R** and run the traditional MRP algoritm to form the E vector for that part. We put the vector in the appropriate row of a temporary **E** matrix. The requirements for this part correspond directly to needed manufacturing operations and thus are accumulated into a **T** matrix.

We next for an intermediate result **X**. Our **X** matrix holds the deadlines for the subassemblies needed to form

Table 2: Matrices used by Enhanced MRP Algorithm

Entity	Size	Description
F_v	$i*(i+l)$	F_v matrix, described previously.
S_v	$i*i$	S_v matrix, described previously.
R	$i*h$	Requirements; number of parts needed at each time unit.
E	$i*h$	Temporary matrix; holds planned order release for a single part.
X	$(i+l)*h$	Temporary matrix; holds subassembly requirements for a single part.
T	$i*h$	Assembled parts; shows how many of each part must begin assembly.
L	$l*h$	Purchased parts; shows how many of each part must be ordered.

our current part. Manufactured subassemblies form the first i rows of **X** and are added into the **R** matrix. Purchased subassembies form the remaining l rows of **X** and are accumulated into the **L** matrix.

When the loop finishes, **T** holds the number of subassemblies which must begin manufacturing at each time unit. The **L** matrix at this point holds the due times of purchased parts; to account for lead-times and possible maximum order sizes, we run the traditional MRP algorithm one more time on each row of **L**.

```
for n = 1 to i
  E = 0
  run MRP on row n of R; set row n
     of E to resulting E vector
  T = T + E
  X = (Sv * Fv)T * E
  R = R + interior nodes of X
  L = L + leaf nodes of X
```

Figure 10: Matrix MRP Algorithm

Conclusions

We have integrated the representations used by machine planners, assembly trees, and MRP systems. This mapping allows machine planners to be easily integrated into manufacturing systems and allows for the execution of short-term plans and the understanding of long-term plans.

We have shown that assembly trees in use among manufactures can be used to encode plan operators. The output of a planner can be converted into two matrices which encode operating constraints in a manner most convenient for an existing real-time manufacturing control system. For real-time systems, we have shown how we can easily form resource useage matrices based on the resulting plan.

Over longer planning horizons, we have shown how the same two matrices, along with an abstract notion of limited production capacity, can be used in a modified MRP system. This allows manufacturers to detect potential scheduling problems well in advance and seamlessly integrates assembly trees, capacity requirements planning, manufacturing resource planning, and symbolic planning.

Future work includes integrating a scheduler into the system to automate the formation of F_a, and to further expand the MRP algorithm to automatically determine the capacity constraints for the various subassemblies and manage interacting constraints.

Acknowledgements

This research was supported in part by the National Science Foundation, under grant GER-9355110.

References

Baker, K., editor. 1974. *Introduction to Scheduling and Sequencing*, John Wiley and Sons, New York.

Bedworth, D., and Bailey, J., eds. 1987. *Integrated Production Control Systems.* John Wiley & Sons.

Ghosh, S; Hendler, J; and Kambhampati, S. 1992. *Nonlin Common Lisp Implementation (v1.2) User Manual*. Computer Science Dept, University of Maryland.

Gracanin, D.; Srinivasan, P.; & Valavanis, K. 1994. Paramertized Petri-Nets and their Application to Planning and Coordination in Intelligent Systems. *IEEE Transactions on Systems, Man, and Cybernetics.* 24(10): 1483-1497.

Harris, B.; Cook, D.; & Lewis, F. 2000. Automatically Generating Plans for Manufacturing. to appear in *Journal of Intelligent Systems*.

Harris, B.; Lewis, F.; & Cook, D. 2000b. A Matrix Formulation for Integrating Assembly Trees and Manufacturing Resource Planning (MRP) with Capacity Constraints. Forthcoming.

Kusiak, A. 1992. Intelligent Scheduling of Automated Machining Systems. in *Intelligent Design and Manufacturing*, New York: Wiley.

Tacconi, D., and Lewis, F. 1997. A new Matrix Model for Discrete Event Systems: Application to Simulation. *IEEE Control Systems Magazine* 62-71.

Wilkins, D. E. 1990. Can AI Planners Solve Practical Problems? *Computational Intelligence* 6(4):232-246.

Wolter, J; Chakrabarty, S; & Tsao, J. 1992. Methods of Knowledge Representation for Assembly Planning, *Proceedings of NSF Design and Manufacturing Systems Conference*. 463-468