Deadlocking in Material Handling Systems

Richard C. Wilson

The University of Michigan
Department of Industrial and Operations Engineering
Ann Arbor, Mi. 48109

Technical Report 85-13

Deadlocking in Material Handling Systems

Richard C. Wilson
The University of Michigan
Department of Industrial and Operations Engineering
Ann Arbor, MI. 48109

Closed material handling systems can be designed with protocols which lead to deadlocks, where further movement of the system is impossible without external intervention. Deadlock states also can lead to premature termination of simulations. Three examples of material handling systems which can deadlock are described. Models for identifying systems with deadlocking potential are described and evaluated.

Many automated material handling systems can be viewed as closed systems with limited space for material in queues or in transit, and captive carts or fixtures which circulate within cyclic routes. Because of the associated costs, systems are designed with minimal numbers of carts, fixtures, and waiting spaces. An unexpected consequence may be the occurence of deadlocks comparable to "gridlock" in New York City. In spite of the potential disruption which deadlocks generate in the operation of automated handling systems, the subject has received little attention in its material handling context.

One purpose of this paper is to describe this problem by means of three real instances in which deadlocking can occur. We provide a survey of methodologies for analyzing the detection or avoidance of deadlocks in existing systems, and for designing deadlock free systems. We also describe two models developed specifically for deadlock detection in material handling systems.

Deadlock-potential adds to the cost of automated handling systems in several different ways. Larger queue space may be needed to reduce the chances of deadlocks. More elaborate look-ahead rules may be programmed in the controls to avoid potential deadlock states. Throughput may be reduced and lead times extended as a result of the time lost in relieving the deadlock. For example, in an automated guided vehicle system (AGVS), a manually operated lift truck may be needed to unload a blocked automated guided vehicle (AGV) so that the AGV is freed to move a load from a blocking output queue to an available input queue. It may not be obvious however which AGV to unload and to which blocking queue it should be dispatched.

Simulation is the technique most appealing for analysis of the capacity of a handling system. Unfortunately, unless the real system is guaranteed to be deadlock-free, a valid simulation model itself may deadlock in the early stages of the simulation run. Therefore, it may be impossible to obtain any data on system capacity. Furthermore, even if a simulation encounters no deadlock state, deadlocks may occur when the real system begins operation. Future changes in the product-mix or work load may introduce deadlocks into an operation which previously seemed deadlock-free.

Examples of Material Handling Systems

A simulation study of heuristic rules for dispatching automatic guided vehicles (AGV's) in a job shop is described in Egbelu and Tanchoco (1981). The shop consists of thirteen departments containing multiple identical machines in each department. Each department except the receiving and shipping departments has a single capacitated input and output queue. Jobs move in unit loads on six AGV's on a route defined by the nature of the job. At each department, the delivering loaded AGV transfers its unit load to the input queue, if space is available. Unit loads are drawn from the input queue and processed by the machines in the department. When finished, loads are discharged into the output queue, if space is available. Loads wait until an empty AGV arrives to remove the processed load and carry it to the next department on its routing. If the output queue is full, machines are not able to discharge completed work and thus become blocked. Incoming work continues to fill the incoming queue until it is unable to accept additional loads. An arriving AGV is unable to discharge its load and therefore remains unavailable to move completed loads from output queues to the next department on their routings. Thus deadlock may occur if full output queues cannot be emptied because all carts are unable to discharge their loads, or are blocked from reaching pick-up points due to interference from other carts. When they encounter this difficulty in their simulation, Egbelu and Tanchoco set the queue capacities to infinity. Unless large queue space is a realistic alternative in practice, however, the performance measures from such a simulation are questionable indicators of the performance of the real system.

As a second example, consider the conveyor transfer scheme in figure 1 which was part of a comprehensive material handling system proposed for a high volume manufacturing facility. The transfer scheme was intended to provide an interface between two cart-type material handling systems. The transfer receives empty racks which are entering the plant stacked either two or three high on tow-line carts (line B). The stack of empty racks is removed from the tow-cart at transfer station 3. The emptied tow cart then continues to the next transfer station at 5. There, it receives a stack of two or three-high full racks which are coming from the manufacturing plant on line A for shipment. Line A is a track system which carries cars mechanically powered for movement independent of each other. The cars can accumulate in queues anywhere. Cars incoming at A may be empty, or carrying full racks of one type of product stacked two-high, or full racks of a different product stacked three-high. All full racks are transferred at 5 to line B by an automated transfer. Emptied cars continue on their track to receive a stack of empty racks from the tow line at transfer mechanism at 3. A mixture of empty cars and cars with empty racks continues on the tracks to the destacker machine at 1. Here the racks on top are lifted off, and the car now with one rack proceeds to a rack checking activity and subsequently into the plant. The next incoming car on the track must be empty so the destacker can place the bottom rack being held from the preceeding stack on the empty car. Thus, a car entering the destacker with three empty racks must be followed by two empty cars. Furthermore, all cars which leave the transfer are either empty or carry exactly one rack.The sequence of rack/car loadings entering the transfer from the plant on line A is controllable by the shipping department. The sequence of loadings on the incoming towline at B is largely random since it reflects the reverse order in which the racks are loaded into rail cars at the customer plants. It is easy to see that this transfer can deadlock if a cart with empty racks is blocked in transfer station 3 because no empty car is available into which it can be unloaded, and a car with full racks is blocked in transfer station 5 because no empty cart is available into which its full racks can be loaded. If this situation arose in practice, a lift truck was to temporarily unload the empty racks from the cart in transfer station 3 so that it could move to station 5 to receive the full racks.

The third example shown in Fig. 2 illustrates deadlock problems in an unusual automated plating system controlled by programmable controllers. The system consisted of approximately sixty cleaning, plating, and rinse tanks serviced by a battery of automatically controlled overhead monorail hoists. Parts for plating are manually loaded into barrels or racks at an input/output workstation. The hoists carry a barrel or rack to the next tank specified by the plating recipe for the batch of parts. The barrel/rack is placed in the tank for a time not less than the required processing time, and the hoist is free to execute the next pickup and move according to a dynamic priority. The system is planned to be able to process an arbitrary mix of parts in any sequence without operator intervention. Deadlocks arose during the simulation of the system when all available circulating barrels/racks were blocked because the subsequent destination tanks on the recipes were occupied. Trial and error efforts to redesign the tank configuration or the recipes to eliminate deadlocks proved to be difficult.

A fourth example (Fig.3) illustrates an automotive engine test system. Engines to be tested are each loaded on a pallet which circulates on a loop power and free conveyor track. The pallets are encoded so that they can be directed to the first free engine test stand. If all stands are occupied, they continue to circulate until a stand becomes free. The engine and pallet enter the free test stand, and undergo the prescribed tests. When finished, they are coded to travel to the a repair station or to the unload point. An engine automatically exits from the loop on its pallet into the free repair station. After repair, the pallet with the engine is coded for a test station and re-entered onto the conveyor loop. Although this system superficially resembles a simple AGV system, it can never block. The number of pallets in the system is never sufficient to completely fill the loop conveyor. Since the conveyor serves as a queue for all the test and repair stations, the queue while finite, is never filled. Therefore the system can never block, or deadlock.

Prior Research

Egbelu and Tanchoco (1981) describe their encounter with deadlocking in their simulation analysis of a hypothetical AGV system. They offer pragmatic reasons why several of their dispatching rules encountered deadlocks, but provide no analytic approach to deadlock detection or avoidance.

In computing systems, the problem of deadlocking was described as early as 1961 in an IBM technical report. Coffman et al (1971) survey the literature on deadlocks through 1970. We briefly summarize the most pertinent ideas published since, in order to establish notation and definitions appropriate to a material handling context. Deadlock issues arise in a system consisting of a fixed number m of types of resources. In a material handling system a resource is the place occupied by a unit load which moves through the system. Thus a resource may be a machine, a waiting space, or a unit load transport cart. Initially we consider only unit loads; thus assembly, disassembly, or batching are not incorporated in these models. The capacity (holding space) of a resource type i is $t_i <$, a fixed integer. A part in process seizes and releases one or more resources as it traverses a specified route through the system. Thus at each part-step j of its route, the part-step "claims" one or more units of the available resources. C(i,j) is the maximum number of resources of type i required by a part-step j at one time. We assume that C(i,j) can be determined from the operation routings. A(i,j) is the number of resources of type i which are currently allocated to processing part-step j at a particular time. If $A(i,j) < C(i,j)$, the processing step is blocked until sufficient additional resources become available to meet the claim. R(i,j) is the number of units of resource type i requested next by the part-step j, given its current allocation A(i,j). Thus if we know A(i,j), we can determine R(i,j), the resources required at the next step on the part routing. A number of relationships are self-evident. For feasibility the resources required at any step cannot exceed the capacity provided:

$$C(i,j) \leq t_i \qquad i=1,..m;\ j=1,..,n. \qquad (1)$$

The number of resource units allocated to a part step cannot exceed the maximum claim, although a partial allocation is permitted:

$$A(i,j) \leq C(i,j) \qquad i=1,..m;\ j=1,..,n \qquad (2)$$

The total number of resources allocated cannot exceed the capacity:

$$\sum_{j=1}^{n} A(i,j) \leq t_i \qquad i=1,...,m \qquad (3)$$

For example (Habermann, 1969) for the claim matrix $||C(i,j)|| = \begin{pmatrix} 4 & 3 \\ 3 & 4 \end{pmatrix}$, the resource capacity vector $||t_i|| = (4,5)$, and a current allocation matrix $||A(i,j)|| = \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix}$, the system is not in a deadlock state, since we can allocate the free resources to satisfy the resource claim needed to process part-step 2. This gives a revised allocation matrix $||A'(i,j)|| = \begin{pmatrix} 1 & 1 \\ 3 & 4 \end{pmatrix}$. Thus part step 2 can finish its operation, and the freed resources reallocated to process part-step 1 next. Alternatively, if the current allocation matrix is $||A(i,j)|| = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$, the free resources cannot be reallocated in any way to satisfy the claim of either part-step. Hence, assuming allocations are irreversible, a deadlock occurs. Coffman (1972) notes that if each of a set of part-steps is allocated to a resource and there is only one resource of each type, a state graph can be used to represent the system state. Each node corresponds to a resource. If a part-step is currently allocated to resource i and requests resource j (i.e., the next step on its process routing after i is resource j), the graph contains a directed arc from node i to node j. A circuit in the graph is a necessary and sufficient condition for a deadlock. For the more general case of several resources of any type and/or multiple types of resources required concurrently by a part-step, Coffman provides the following algorithm to detect a deadlock condition for a given allocation state. Let s_i be the number of resources of type i currently free. Thus:

$$s_i = t_i - \sum_{j=1}^{n} A(i,j). \qquad (4)$$

The algorithm proceeds as follows:

Step 1: Initialize $w_i = s_i$ for all $i=1,\ldots,m$. "Mark" all part-steps j which are unallocated (eg. $A(i,j)=0$ for all i).

Step 2: Find an unmarked part-step k with $R(i,k) < w_i$ for all i. If one is found, go to step 3; otherwise stop: a deadlock exists.

Step 3: Set $w_i = w_i + A(i,k)$ for all i. "Mark" part-step k and return to step 2.

The algorithm reveals a deadlock by accounting for all ways of completing the allocated part-steps. A deadlock exists if and only if there are unmarked part-steps at termination; i.e. one or more part-steps cannot be completed. Modifications are proposed to improve the computation time in special cases. Similar ideas are developed and extended by Holt (1972), including testing deadlock-freedom for systems with single unit resources ($t_i=1$ for all i) or with a single resource type ($m=1$).

Kameda (1980) describes an efficient algorithm for determining the deadlock-freedom of a system which does not require a deadlock detection algorithm at each state of the system. The procedure requires that the parts (and hence the part-steps) to be admitted to the system be fixed in advance, and their resource requirements be specified. For the same model, Ibaraki et al (1983) provide an algorithm for designing deadlock free systems by selecting the t_i's which minimize:

$$\sum_{i=1}^{m} d_i\, t_i,$$

where d_i is the cost per unit of resource type i. Ibaraki and Kameda (1982) show that several variants of this problem are NP-complete.

In most material handling systems, the set of parts which may be in the system at a given time may be a sub-set of all possible parts. The total number of parts may be constrained by the number of available fixtures, carts, waiting space, or machine space, but the specific set of parts may be continually changing. In these cases, Kameda's algorithm is not directly applicable. Ibaraki and Kameda (1982) show that if the number of part-steps which are allowed concurrently in the system is limited to K, the system can deadlock for K if and only if:

$$K \leq \sum_{j=1}^{m} \max\{ KC(i,j) - t_i, 0 \} \qquad (5)$$

holds for all i. Thus, in the earlier numerical example, the system can never deadlock if K=1; i.e., if at most one part is allowed in the system, but it can deadlock if K = 2. If there is also an upper bound k_j (>1) on the number of part steps x_j from a class (such as all part-steps which use the same fixture), and if

$$K < \sum_{j=1}^{m} k_j,$$

determining if a system can deadlock is NP-complete.

Obermark (1982) and others have modeled algorithms for detecting a deadlock in a packet switching network where messages must be transmitted among the nodes in order to determine the current state of the network. Complications arise when the state of the system changes during the time required to transmit the messages. Toueg and Ullman (1981) prove that if buffers at each node are of size b and b is greater than the maximum number of part-steps of any part type, several local dispatch rules can control the dispatch of parts so that deadlock will never occur. Bracha (1984) addresses the problem of deadlock detection when resource requirements may be of the AND-OR variety, i.e., a part-step may require several other (AND) part-steps concurrently for its next step, or a part-step may require some combination (OR) of other part steps.

Deadlock Freedom in Material Handling Systems

From the preceding survey, we see that in the most general case, determination of deadlock-freedom requires enumeration of all possible states and testing each state to detect if the state can deadlock. In this section we describe two branch and bound algorithms which seek to enumerate only feasible system states for deadlock detection. We show how the models apply to a variety of material handling systems. Computational experience on several realistic problems is reported.

*Integer Programming Model

We first formulate a zero-one integer programming problem which seeks to maximize the number of part steps:

$$z^* = \sum_{j=1}^{M} x_j$$

subject to constraints on the number of part-step unit load spaces which are available:

$$\sum_{j=1}^{M} C(i,j)\, x_j \leq t_i \qquad i=1,\ldots,m$$

and to other configuration constraints (i.e., each unit load j must be fastened to a specific fixture type k while in the system):

$$\sum_{j=1}^{M} F(k,j)\, x_j \leq b_k \qquad k=m+1,\ldots M.$$

The following assumptions are implicit in the model:

1. Parts move as unit loads throughout the system. Each space

constraint is stated in number of unit loads.

2. Parts are not assembled or disassembled during their route through the system.

3. Each part has a unique routing of steps through the system, with known space requirements at each step. The part steps are indexed in the model in sequence from left to right.

4. Only one of any part-step can be in the system at any time. This can be relaxed by adding columns which duplicate part-steps in the model, or by changing the model to permit x_j's to be any non-negative integer.

5. The last step on a part requests next the resources specified for the first step on the same part. Hence the system will block if the finished part cannot be replaced with the same part.

6. Space resources must be empty before a claim can be satisfied.

7. A part-step may claim multiple amounts of several resources simultaneously.

An overview of the major steps in the algorithm to determine if a system can deadlock is given in figure 4. Given the (0,1) model,the algorithm seeks to find a feasible solution state which contains a maximum number of part-steps. Whenever the (0,1) algorithm finds a feasible state containing at least as many part-steps as any previous state, Coffman's algorithm is used to test for a deadlock. If none is found, the algorithm tests other feasible states until either all states have been implicitly tested or a deadlock is found. Thus for designs with p fixtures, n part-steps, and $n<p$, the algorithm may test $n*...*(p+1)/(n-p)!$ different states. For realistic problems (e.g.,$n=50$ and $p=15$) the number of tests may be very large indeed.

In figure 5, we show a very simple array from the (0,1) model for a system with three part types, the first and third parts having two part-steps, and the second having four part-steps (Problem #1AGV). Three resource types and two fixture types are provided. If part-step 1 is in the state ($x_1=1$), it requests the resources and fixtures of the succeeding part-step 2, whereas the last part-step x_6 of the second part type next requests its first part-step x_3.
This model structure lends itself to a simple heuristic for minimum incremental cost resource expansion when a deadlock state is detected. Assume that resources of type i cost d_i dollars per unit of space. If a deadlock is detected, increment the resource capacities t_i for i=1,..,m by an integer amount which relieves the deadlock at minimum total cost. Repeat the deadlock test for that state, etc. A guaranteed optimal resource expansion can be obtained for a deadlocked state using a dynamic programming formulation. We omit the details.

*Network Model

For the simpler application in which a part-step claims only one unit of one resource at any part step we describe a second model based on a network representation for generating all feasible states of the handling system, subject to a variety of realistic constraints. The network consists of five stages as shown in Figure 6. At the first stage h, one node is used to model each fixture type; at the second stage i, each node represents a part-step; each node at the third stage j represents a resource type, and at the fourth stage k, nodes represent "regional" or shared resources. Enumeration of system states requires the solution of a sequence of maximum flow problems defined as:

$$\text{maximize} \quad x_{ts},$$

subject to flow balance at each node:

$$\underline{A}\ \underline{x} = \underline{0}$$

where $\underline{A}$ is the node-arc incidence matrix of a network, constructed according to the following rules:

1) Connect source node s to each fixture node h with upper bound u_h equal to the maximum number of fixtures of type h circulating in the system. The model assumes that a fixture remains associated with a particular part throughout all of the processing steps and resource utilizations.

2) Connect each fixture node h by an arc to each part-step i which uses fixture type h. Note that a given part may use any one of several different fixture types. These arcs need not be bounded.

3) Connect each part step node i by an arc to the resource type node j on which part step i is processed. During enumeration of states, these arcs provide a matching of part-steps to resources, where the "flow" on arc (i,j) is exactly one if part-step i is in the system for that state, and zero otherwise.

4) Connect resource node j to the appropriate joint use node k (if any) or to the terminal node t, with an upper bound on the arc equal to t_j, the number of resources of type j available.

5) Connect each joint use node k to the terminal t with an arc whose upper bound is the capacity for joint use of the incident resources.

Figure 7 provides a flowchart of the program for testing an arbitrary material handling configuration for deadlock-freedom. The implicit enumeration algorithm is based on Jensen and Barnes (1980). A deadlock state is determined by the algorithm described by Coffmann (1972). Part-step matching to resources is established by a standard out-of-kilter sub-routine appropriately configured for this application. Thus the testing procedure draws on three routines selected because of their ready availability, not because of their efficiency.

Modeling and application of the algorithm is illustrated by a simplified example of a flexible manufacturing system using AGV's. The system has three types of resources: two AGV's, one machine type 1, and one machine of type 2. No queue space is provided at either machine. The two AGV's are identical. Two part-types are processed. Part type 1 is processed on machine 1; part-type 2 is processed first on machine 1 then on machine two. Both parts are mounted on fixtures while in the system, part 1 on fixture A and part 2 on fixture B. Two fixtures type A and one of type B are provided. Movement between and to/from the machines requires use of either AGV. Schematically the layout might be as shown in figure 8a. However the explicit details of the layout are not relevant to deadlock-detection, since interference between AGV's, travel rates, and throughput capacity are not considered. The network model for this system is shown in figure 8b. The algorithm detected a deadlock in the first state evaluated which has a part 1 on an AGV requesting machine 1, another part 1 being processed on machine 1, and a part 2 on the other AGV, also requesting machine 1. An alternate of this same system (Problem #2AGV) with three AGV's, 2 type 1 machine spaces, and three each of both fixtures is found to be deadlock-free. A second trivial variation of the original system with no fixture type A, generated the deadlock enumeration tree in figure 9.

A toy example of the automated plating system provides a second example of the network model for deadlock detection. We assume there are six types of plating tanks. There are two identical tanks of type 1, and one of each of the others. Parts to be plated are loaded into either a plating "barrel" or a rack. Two of each are available. Three different plating recipes are assumed. Recipe 1 requires parts in a barrel to cycle through tanks 1, 2, and 5 and then back to a load/unload station. Recipes 2 and 3 require racks, and cycle through tanks 1, 3, 4, and 5 or 1, 2, and 6 respectively. The network model of this system (Problem #3TNK) is shown in figure 10. A deadlock state found by the algorithm is shown in figure 11. This deadlock was found in the seventh state evaluated.

Computational Experience

A model of the real plating system consisting of 64 tanks and 16 part types with a total of 270 part-steps was run on the network algorithm. Computation was halted after 120 CPU seconds and over 4000 feasible states had been tested. Determining if the system were deadlock free could require examination of all combinations of 17 fixtures among 270 part steps. Several sub-sets of part types were tried with a reduced set of resource types. The deadlock shown in Figure 12 was detected in sub-problem #7TNK with 19 tanks, 21 part-steps and 17 fixtures. Computation times for this and other smaller problems are shown in Table 1. Further reductions in CPU time for the network algorithm could be achieved by simplifying the detection algorithm for the single resource use by each part-step.

Conclusions

We described the development of programs which detect deadlock potential for a wide class of material handling systems designed for automated manufacturing. For large problems, the CPU time of these is likely to be prohibitive. Clustering methods for selecting subsets of part-steps with high potential for deadlocking might be devised to reduce problem size. A number of other questions are yet to be resolved. For example, the models do not incorporate the contribution to deadlock which might be caused by interference between carts. Furthermore, we have not developed Bracha's concepts for AND-OR detection models to permit the modeling of part steps which include assembly or dis-assembly. As a consequence, we have not modeled the transfer system described earlier.

In addition, we have barely touched on the issues of deadlock-avoidance and design. With carefully devised sequencing rules, potential deadlock states can be avoided, as Egbelu and Tanchoco indicate in their AGV simulations. However, a systematic methodology for devising rules which avoid deadlocks in an existing system is needed. For example, if the model deadlocks because the last part-step requires that it be replaced by the first part-step for the same part-type, resource requests of the first steps of alternate part-types might be considered, particularly part types which might be accepted without deadlocking or violating constraints. These rules might also be useful in selecting parts to load into a system to avoid deadlocking. The rules for choosing among several part types candidates however may impact the long term proportion of part-types accepted, or the prospect of future deadlocks. The design of minimum-cost deadlock-free systems has been reported by Irabaki (1983). However, this work only applies to the case where the the number of part-steps in the system is fixed and known in advance. For material handling applications we previously noted that the part-steps may change during operation, so that Irabaki's algorithm needs extension for material handling applications.

References for "Deadlocks in Material Handling Systems"

Araki, T., Y.Sugiyama, T.Kasami, J. Okui; "Complexity of the Deadlock Avoidance Problem", _Proc. 2nd IBM Symp. on Mathematical Foundations of Computer Science_, IBM Japan, Tokyo (1977) p.229-252.

Bracha, Gabriel, Sam Toueg; "A Distributed Algorithm for Generalized Deadlock Detection", Tech.Report TR 83-558, Dept. of Computer Science, Cornell University, Ithaca, NY 14853 (July, 1984).

Coffman, E.G.Jr., M.J.Elphick, A. Shoshani; "System Deadlocks", _Computing Surveys_, 3:2 (1971) p.67-78.

Egebelu,Pius J., J.M.Tanchoco; "Characterization of Automatic Guided Vehicle Dispatch Rules", _Int.J.Prod.Res._, 22:3 (1981) p.359-374.

Garey,M.R., D.S.Johnson; _Computers and Intractability_, W.H. Freeman & Co. San Francisco, CA 94104. (1974)

Gold, E.M., "Deadlock Protection:Easy and Difficult Cases",_SIAM J. Comput._, 7 (1978) p.320-336.

Haberson,A.N., "Prevention of System Deadlocks", _Communications of ACM_, 12:7 (July,1969) p.373-385.

Holt, Richard C., "Some Deadlock Properties of Computer Systems",_Computing Surveys_, 4:3 (September,1972) p.179-195.

Ibaraki,T., T.Kameda; "Deadlock-Free Systems for a Bounded Number of Processes", _IEEE Transactions on Computers_, C-31:3 (March,1982) p.188-193.

Ibaraki,T., H. M. Abdel-Wahab, T. Kameda; "Design of Minimum-Cost Deadlock-Free Systems", _Jour. A.C.M._, 30:4 (Oct.1983) p.736-751.

Jensen,P., Barnes, _Network Flows_, Wiley & Sons, New York,N.Y. (1980).

Kameda,T.; "Testing Deadlock-Freedom of Computer Systems", _Journ.ACM_, 27:2 (April,1980) p.270-280.

Reingold,E.M., J.Nievergelt, N.Deo; _Combinatorial Algorithms:Theory and Practice_, Prentice-Hall, Inc, Englewood Cliffs, N.J. 07632 (1977).

Silberschatz, A., Z.M.Kedem; "A Family of Locking Protocols for Database Systems that are Modeled by Directed Graphs", _IEEE Transactions on Software Engineering_, SE-8:6 (November,1982) p.558-562.

Toueg, Sam, Jeffrey D. Ullman; "Deadlock-Free Packet Switching Networks", _SIAM J. Comput._, 10:3 (August, 1981) p.594-611.

Wilson, Richard C. Conveyor Transfer Analysis:A Case Study, Working Paper #2, Manufacturing Flow Research Project, Department of Industrial and Operations Engrg., The University of Michigan, Ann Arbor, MI 48109 (1977).

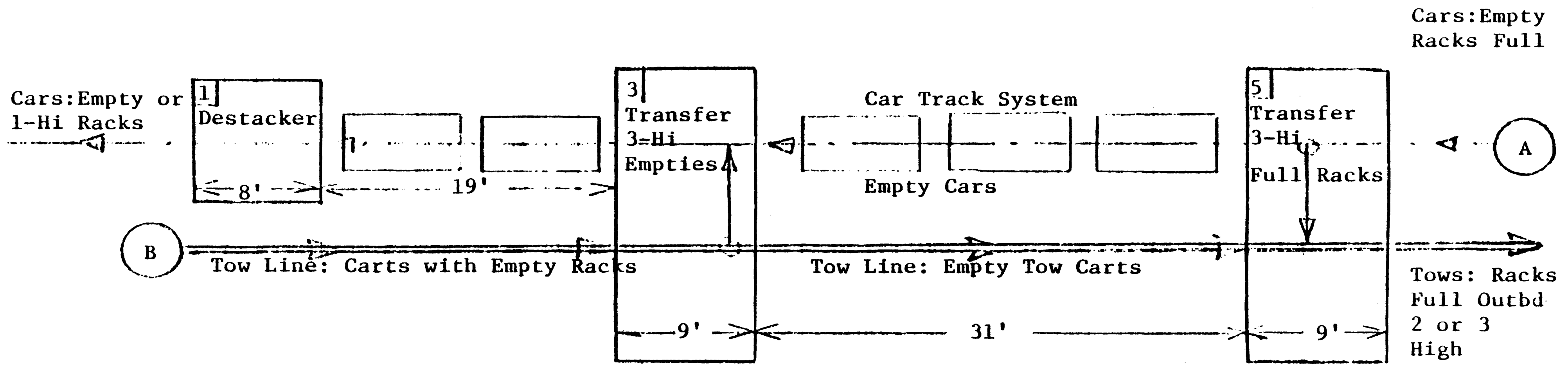


Figure 1. Plan View of Transfer Scheme. Scale 1 inch=10 feet.

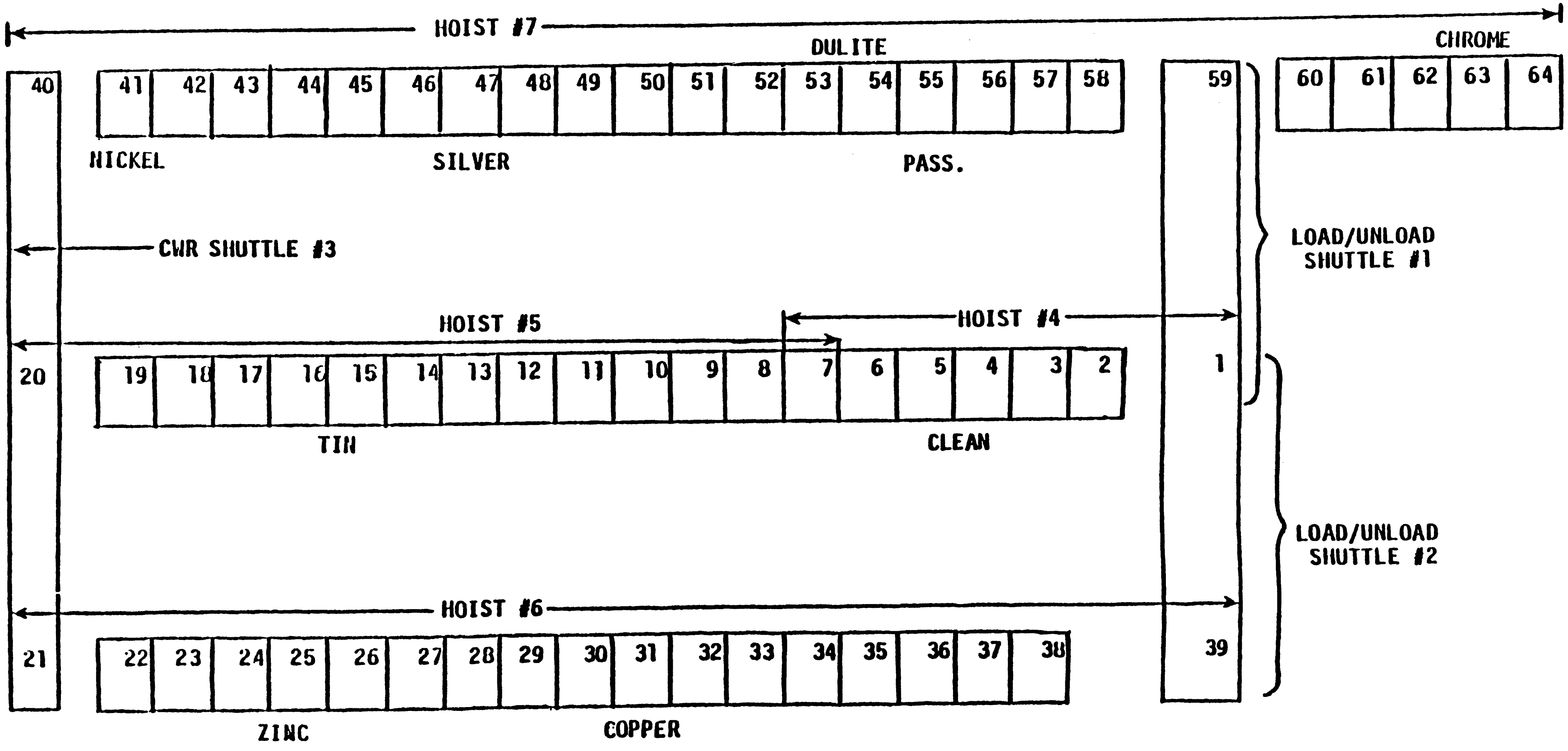


Figure 2. Layout of tanks and hoists in automated flexible plating system.

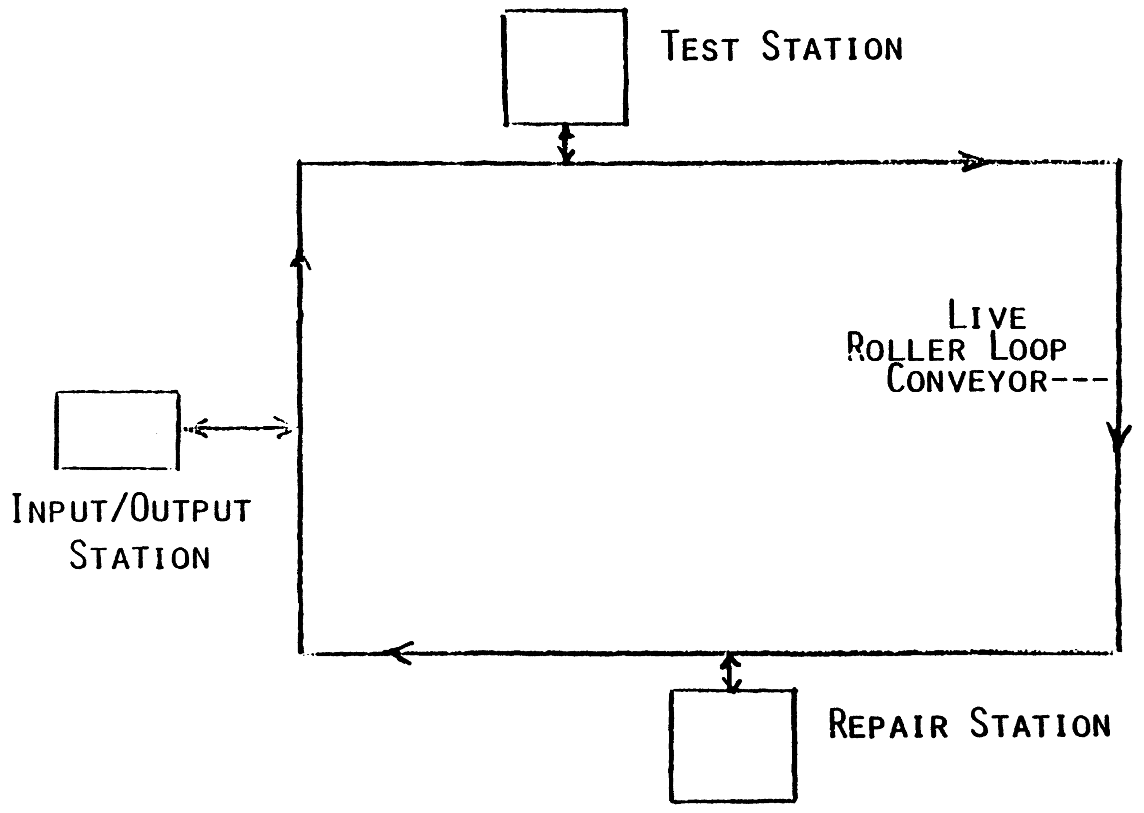


FIGURE 3. AUTOMATED ENGINE TEST CELL

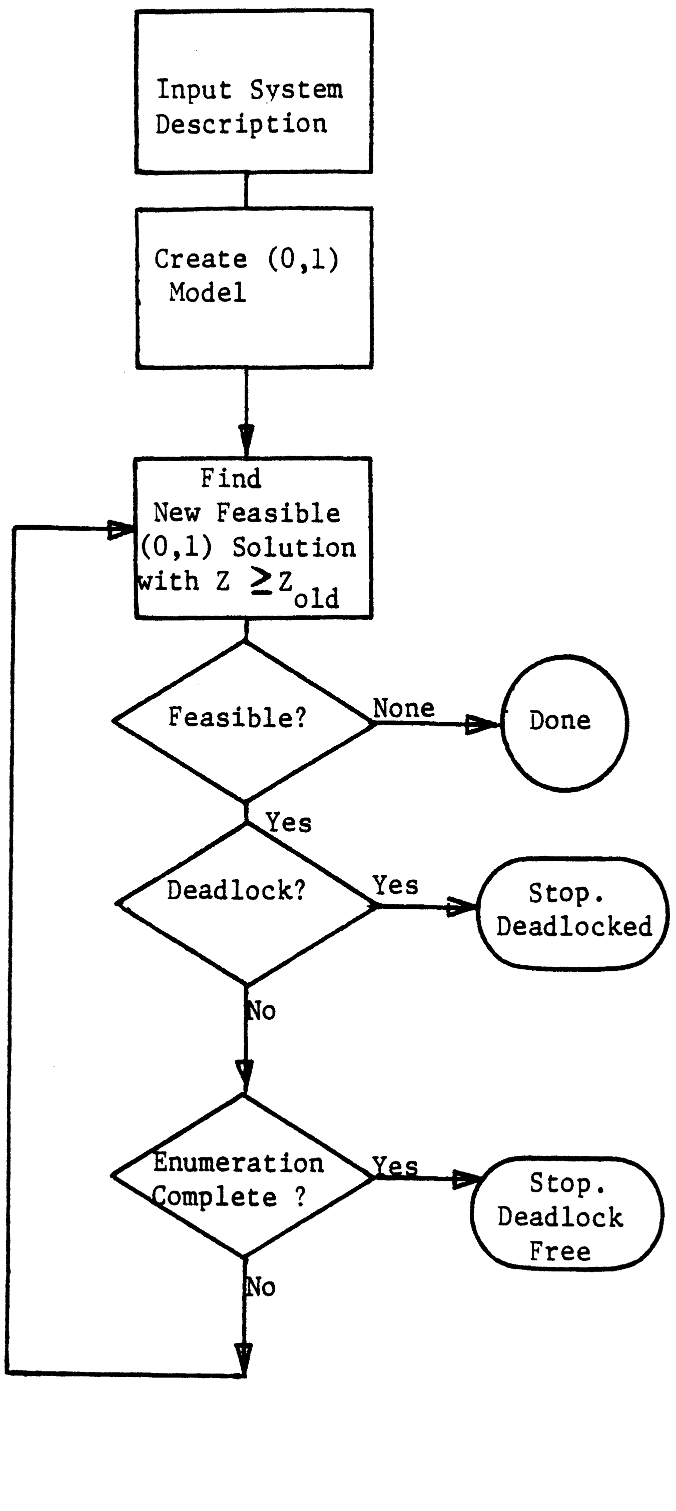


Figure 4. Flowchart of (0,1) model for deadlock detection

Part #1		Part #2				Part #3				
$x_1 \rightarrow$	x_2	$x_3 \rightarrow$	$x_4 \rightarrow$	$x_5 \rightarrow$	x_6	$x_7 \rightarrow$	x_8			
1	0	1	0	1	0	1	0	$\leqslant$	3	Resource #1
0	1	1	1	0	0	0	1	$\leqslant$	1	Resource #2
2	1	0	0	0	1	2	1	$\leqslant$	6	Resource #3
1	1	0	0	0	0	1	1	$\leqslant$	2	Fixture #1
0	0	1	1	1	1	0	0	$\leqslant$	1	Fixture #2

Figure 5. Array for Toy Example (Problem #1AGV) of (0,1) Model

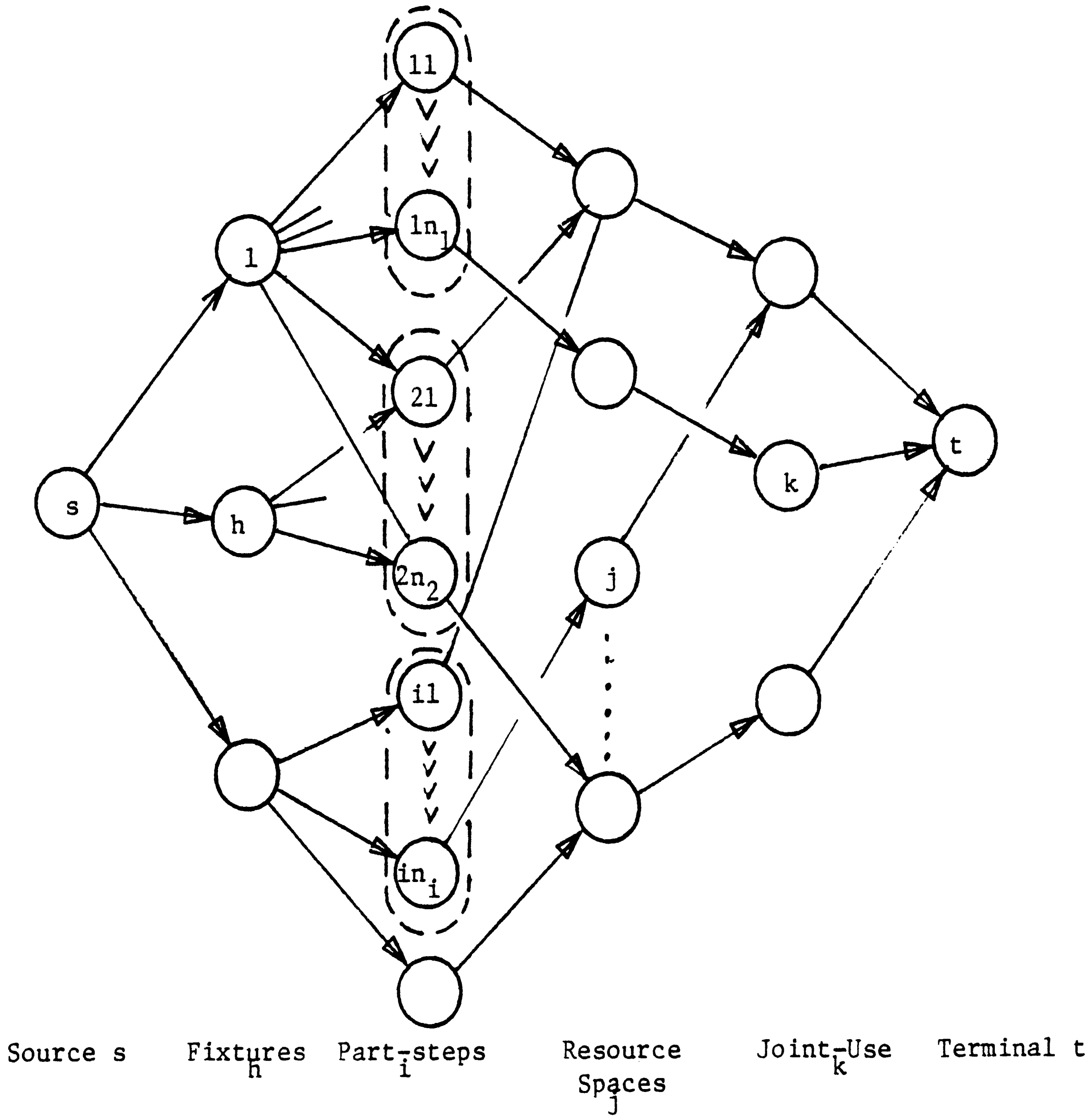


Figure 6. Stages in a representative Network

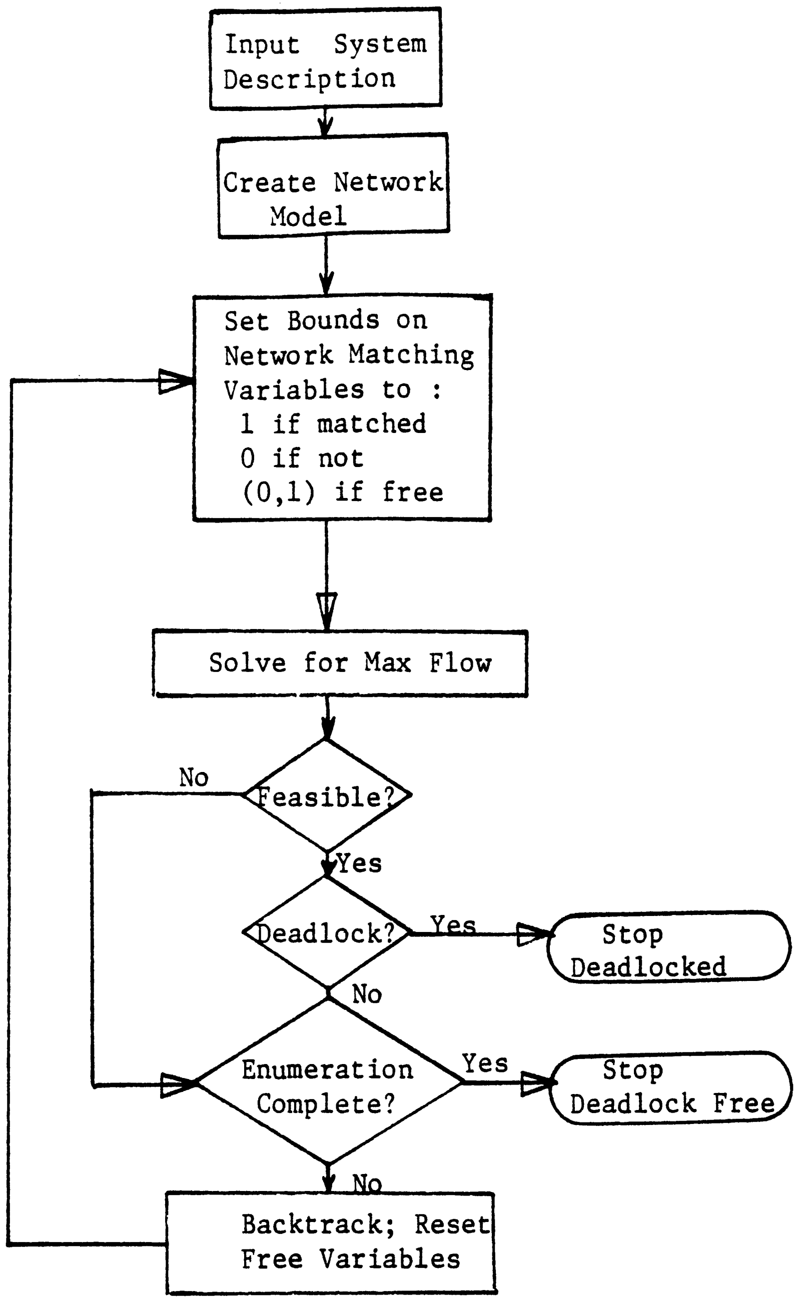


Figure 7. Branch and Bound/Network Algorithm for Deadlock Detection

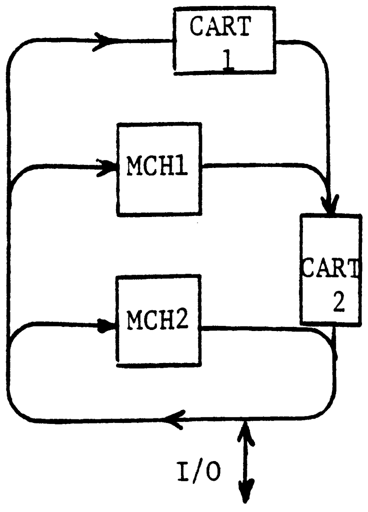


Figure 8a. Simple FMS with two AGV's (CART) and two Machine types

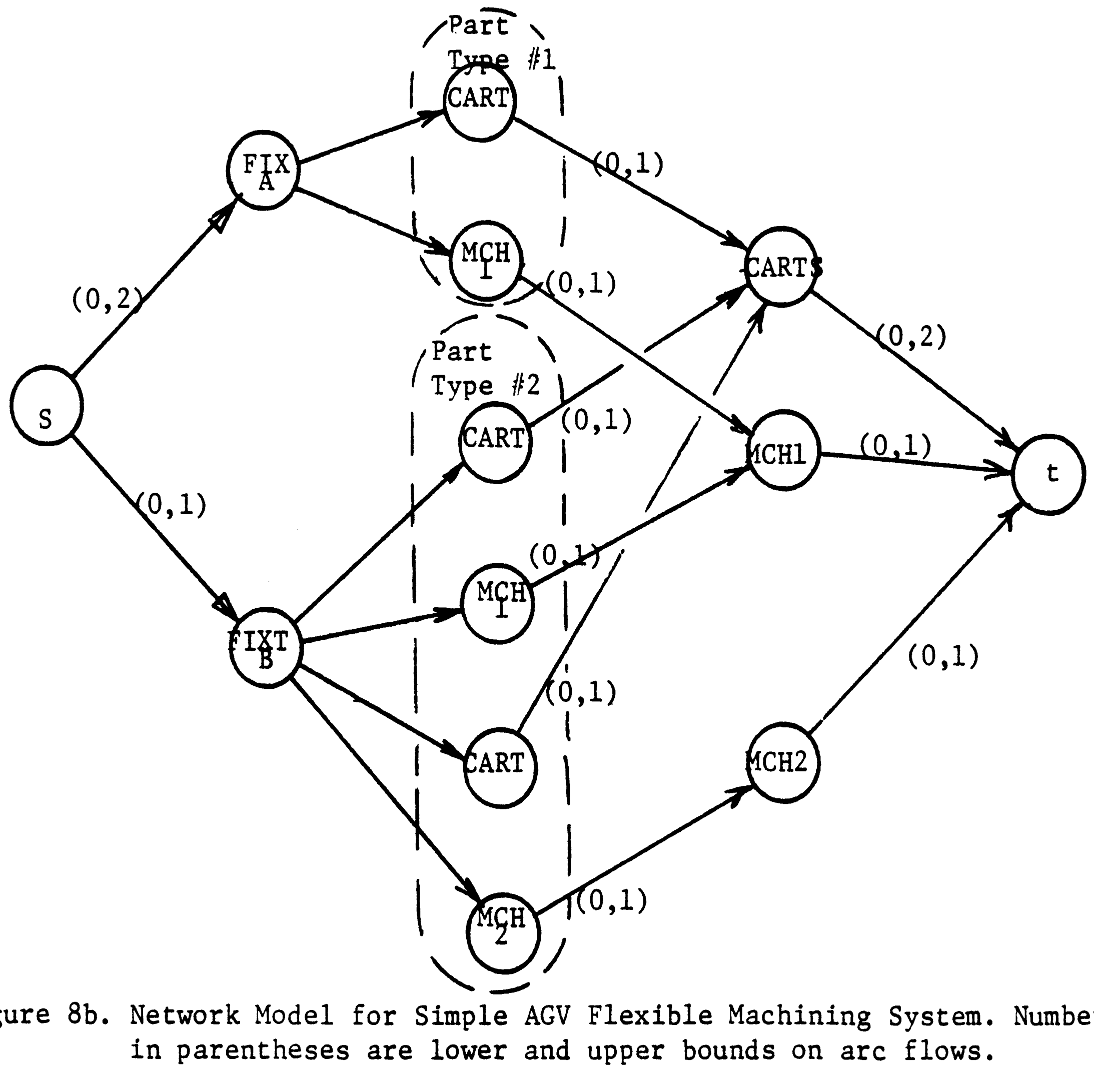


Figure 8b. Network Model for Simple AGV Flexible Machining System. Numbers in parentheses are lower and upper bounds on arc flows.

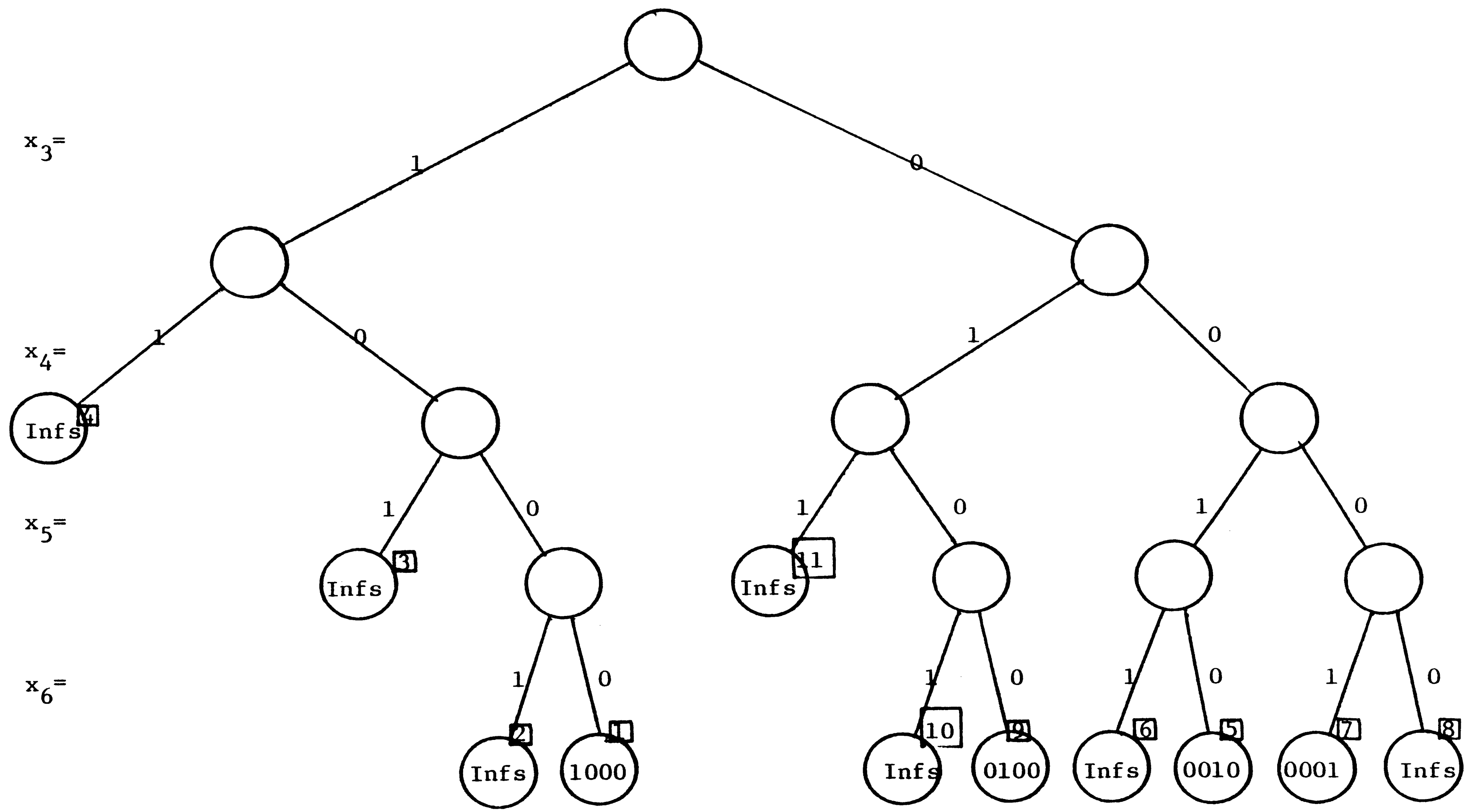


Figure 9. Branch and Bound Enumeration Tree for Simple AGV Flexible Machining System in Figure 8 with no fixture type A. "Infs" = infeasible. Branching values of (0,1) are given on edges for x_3, x_4, x_5, and x_6. Values of x's are in nodes. Numbers in □ are the sequence of branchings.

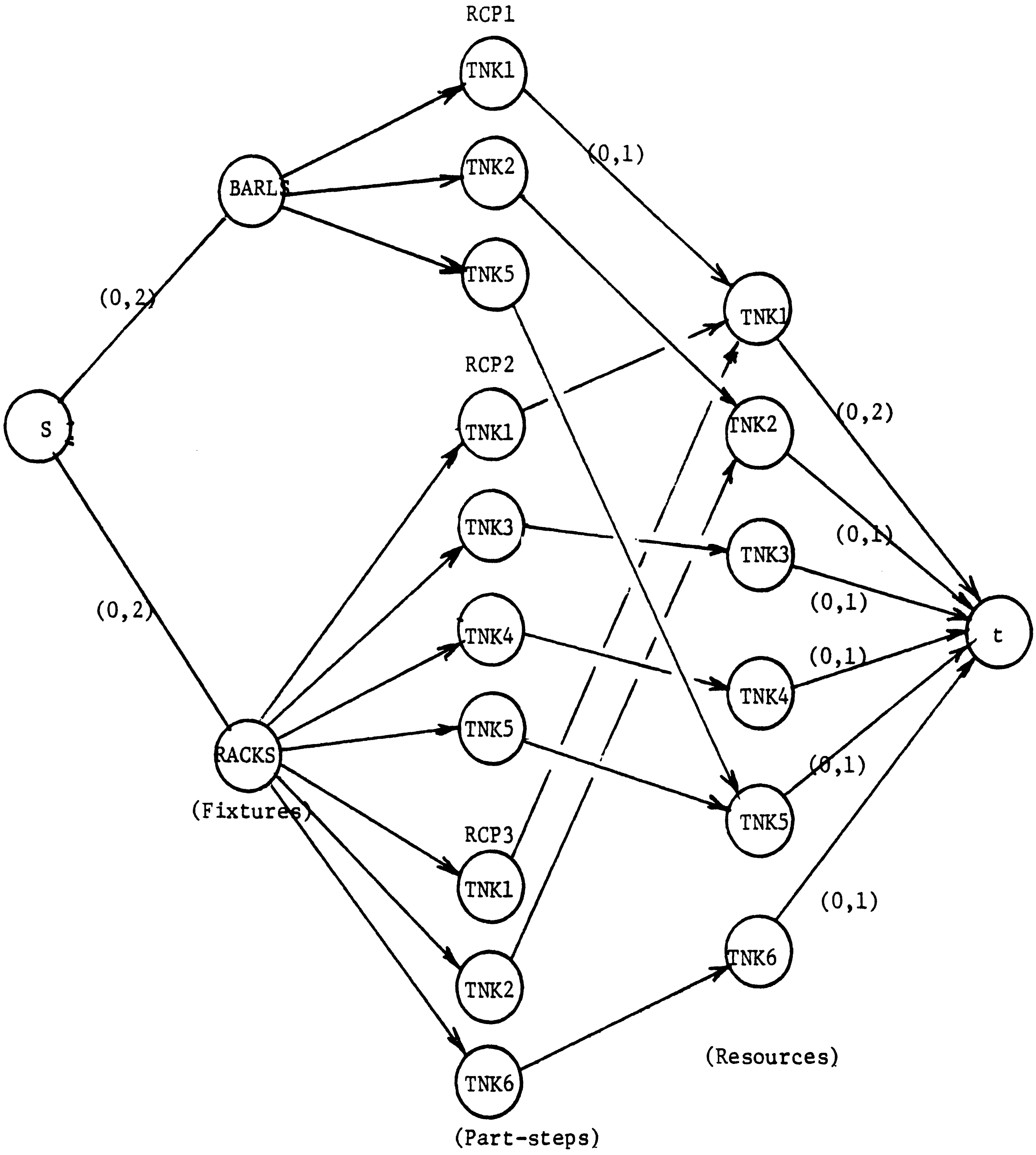


Figure 10. Simple Example of Network for Automated Plating System

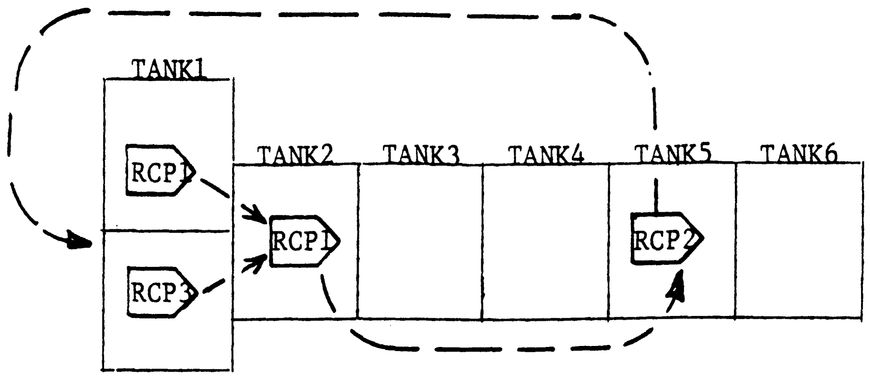
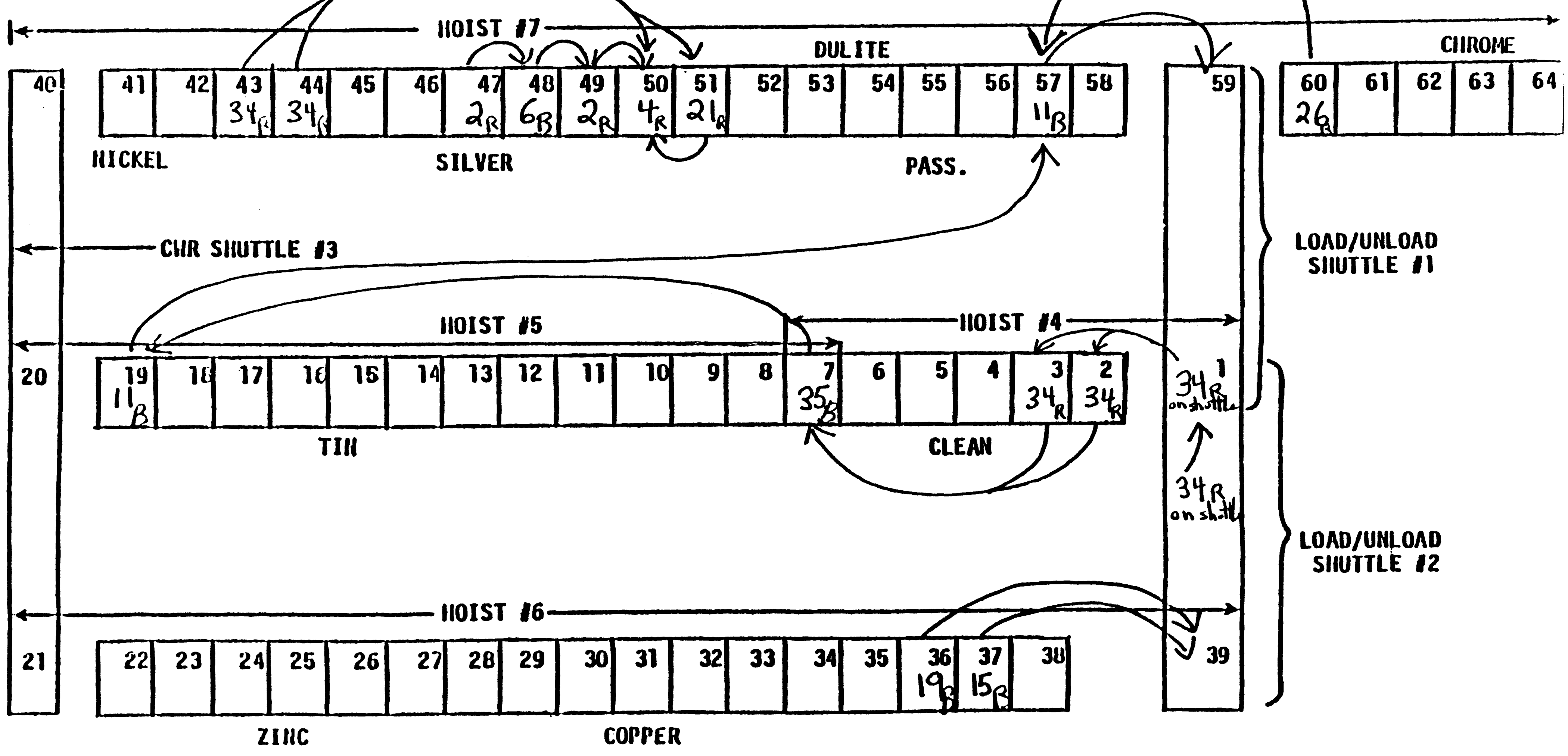


Figure 11. Deadlock State for the Simple Example (Problem #3TNK) of the Automated Plating System. Recipes (part-types) numbers are in . Arrows indicate next requested tank type.

A SAMPLE DEADLOCK CONFIGURATION

(11 RACKS, 6 BARRELS)



NOTE: AS FEW AS FOUR RACKS AND THREE BARRELS CAN PRODUCE A DEADLOCK OF THIS CONFIGURATION.

Figure 12. Deadlock State for Real Automated Plating System (Problem #7TNK)

PRELIMINARY COMPUTATIONAL RESULTS

PROB#	# FIXT.	M	N	(0,1) MODEL #DLK TESTS	(0,1) MODEL CPU ML SEC	--NET MODEL------ #NETS	--NET MODEL------ #DLK TESTS	--NET MODEL------ CPU ML SEC	RESULT
1AGV	3	3	8	5	25	11	3	47	DDLOCK
2AGV	6	3	6	15	38	6	4	34	FREE
3TNK	4	6	10	60	187	8	7	53	DDLOCK
7TNK	15	19	21	4250	?	201	76	1786	DDLOCK
7TNK	17	19	21	130	22122	56	20	475	DDLOCK
8AGV	5	3	11	11	39	1	1	35	DDLOCK
9AGV	5	3	11	140	480	164	120	414	FREE

Table 1. Comparative Computation Times and Detection Iterations for (0,1) and Network Models. CPU time in milli-seconds. M=number of resources; N= total number of part- types.