Distributing empty freight cars throughout a railroad system in anticipation of future requirements is an allocation problem. The actual movement of cars can be examined in terms of a space-time diagram. An inductive network flow algorithm for solving this problem utilizing the network underlying the space-time diagram is developed and illustrated by an example.

A computer program implementing this algorithm is discussed, along with the context in which it might be used. Possible extensions are also presented.

A network algorithm for empty freight car allocation

by W. W. White and A. M. Bomberault

Some of the earliest applications of operations research techniques were made in the field of transportation. Until recently, however, little was done within the railroad industry. This paper describes an application of network flow analysis to one problem of that industry: empty freight car allocation. Other modes of transportation may benefit from the same analysis and the algorithm developed herein.

The allocation problem is that of distributing empty freight cars throughout a railroad system in anticipation of future requirements. This allocation should be accomplished so as to minimize the cost of moving the cars into position from the locations where they become available.

The movement of the empty cars is examined in terms of a space-time diagram, which schematically represents the various paths that cars may travel to reach their proper destination at a specified time. The "essential structure" of the space-time diagram is used to construct a corresponding network on which car movements are interpreted as flows over arcs from node to node.

When attention is focused on a single car fleet (in which all empty cars are assumed to be interchangeable), the allocation problem can be mathematically defined as a transshipment problem[1] represented on the network. The special structure of the network can be used advantageously in developing an inductive algorithm to solve this transshipment problem.

This paper examines in detail the development of the network and the algorithm. As an example of the implementation of the algorithm, a computer program using it is discussed. The paper concludes with a discussion of generalizations and extensions to the algorithm.

The allocation problem

Historically, the distribution of empty freight cars has been handled by dispatchers. Railroad agents, responsible for customer requirements, inform the dispatchers of cars needed by their customers. Empty cars are often supplied to locations on the basis of rate of use, and the dispatcher plans for the allocation of these cars on this basis. If no untoward events occur, this allocation system works fairly well, based as it is upon the experience of the dispatchers.

However, fluctuations in the rate of use are not uncommon, and can throw this system out of balance. If one extra car cannot be obtained from the usual areas of supply, there is a dislocation in the system. Clearly, if this situation is magnified by occurring many times for many customers, matters become too complex for the allocation system to remain reliably effective. Customers are dissatisfied because there is often a shortage in the number of cars they require, or car utilization declines because of unnecessary or overly complicated car movements. Sometimes both results occur.

The agents often build up their own safety buffers of empty cars to draw on if shortages develop. Although this practice can alleviate customer dissatisfaction, it decreases the average utilization per car, reducing the number of cars available to the railroad as a whole, and thus necessitates the purchase or rental of more cars.

Clearly, a more efficient system for the allocation of empty cars would be beneficial. As a prerequisite, a rolling stock information system is necessary, for cars cannot be allocated effectively without knowing what cars are or will be available, where, and at what time. Indeed, many railroads now have or are developing such a system. However, the need remains for a systematic method of employing this information to achieve increased car utilization and/or customer satisfaction.

early allocation methods

Automated methods for allocating empty freight cars have been proposed. An early attempt by Feeney[2] was programmed, but was unsuccessful in its application, apparently because of the complexity of the model and the requirements regarding the necessary supporting data. This particular model incorporated inventory aspects as well as allocation and distribution aspects.

More recently, a simpler linear programming formulation of the allocation problem was developed and implemented.[3–5] This model is essentially a multiperiod transportation problem, and is highly successful in its application. It was designed to minimize

the total time in transit from supply points to demand points. However, it did have more far-reaching effects. Along with decreasing the frequency of car shortages (and the extent of the shortages), the number of cars needed to adequately supply customers was also reduced, and car utilization increased. Indirectly, customers' satisfaction increased because an adequate supply of cars enabled them to cease worrying about an excess final product inventory.

The success of this linear programming model was a spur to the development and implementation of the model presented in this paper. The model developed here uses a network flow type of algorithm, rather than a linear programming simplex method, and is designed for a formulation that allows consideration of intermediate stops between supply and demand points. Thus, in terms of computer operation, a more general problem can be solved faster with less storage. Furthermore, detailed information on the routing of the cars is available automatically, whereas further calculations are required to develop this information using the earlier approach.

Patterns of movement

Empty car movements take place both in space and in time, and a train is the means used to haul the cars. If it is known that a train will go from, say, station A at time t_A to station B at time t_B and then to station C at time t_C, a possible movement for some empty cars is along the same route. The time taken to go from B to C is $t_C - t_B$. The movement can be diagrammed as follows: let time be represented on the horizontal axis, and consider each of the three locations as a point on the vertical axis (in any order); the routing is then indicated by two arrows, one corresponding to the first leg of the journey with its tail at the point (t_A, A) and its head at the point (t_B, B) and the second arrow corresponding to the second leg, with its head at (t_C, C) and its tail at (t_B, B). The diagram of the movement is illustrated in Figure 1.

Figure 1 Space-time diagram of empty car movement

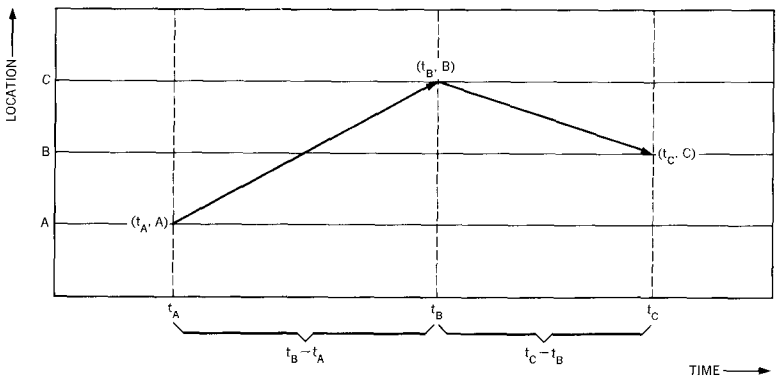
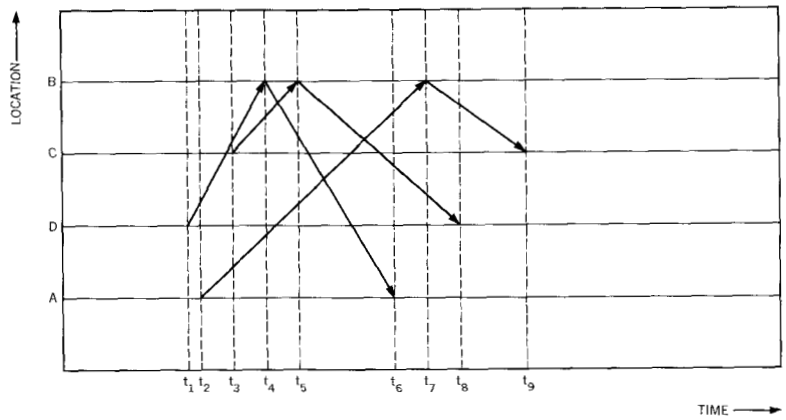


Figure 2 Space-time diagram of three trains and four locations



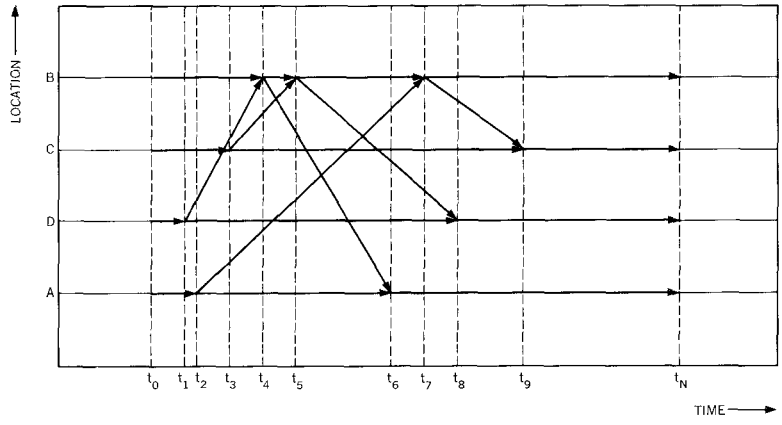
Every train that hauls cars gives rise to a similar sequence of arrows corresponding to possible movements of empty cars from one location to another using that train. If all these arrows are drawn on the same diagram, then a pattern of movement emerges. The horizontal distance from the head of any arrow to its tail indicates the time necessary to complete the corresponding leg of the journey. Figure 2 is an example of a pattern of physical movement arising from three different trains visiting four different locations.

Note, however, that in Figure 2 it is possible for empty cars to go from location D to location C. This trip is accomplished by traveling the first leg of the journey on the train from D to B, then waiting at B from the time the first train leaves until the time the next train arrives and leaves, further waiting at B until the last train arrives, and then traveling the last leg of the journey on this train from B to C. We can look at the waiting period as a purely temporal movement, so that the journey from D to C consists of four legs, with the middle legs representing temporal movement. We can also include in the diagram, therefore, arrows that represent the various possible waiting periods (temporal movements). The space-time diagram of Figure 2 is completed by the addition of these arrows, resulting in Figure 3 (where t_0 and t_N represent the beginning and the end of the time span during which the movements take place).

arrow relationships

The critical features of the space-time diagram center around the relationships between the arrows. The actual time is not really important per se. In fact, the only times at which an arrival and/or departure, i.e., an "event," occurs are those times at which one arrow joins (or leaves) other arrows. For example, in Figure 3, the arrival and departure at location B of the train that goes from C to B to D is an event (with which is associated a time). The crucial relationship between events is in terms of precedence—what events precede and follow a particular event—and this precedence is indicated by the arrows of the space-time diagram. The

Figure 3 Space-time diagram including temporal movements

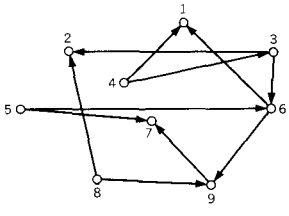


length of time between events is not highly important since it is always possible to record the events and their times. As before, the arrows represent the possible empty car movements that occur. An event can be represented on the diagram by a *node*. Thus each arrow connects two nodes.

It is not necessary to identify each node directly with a location and a time. The essential structure is still kept if each node is assigned a number. The location and time corresponding to any particular node number can then be recorded on a separate list. If it is understood that the series of nodes identified with the same location includes in the first node whatever waiting may have occurred at the location prior to the first connection, and includes in the last node whatever waiting may occur after the last connection, it is sufficient to consider a network with numbered nodes, plus some additional information, when examining possible empty car movements within a give time span. The possible movements correspond to flows over the various arrows of the network. A network containing the essential structure of the space-time diagram in Figure 3 appears in Figure 4, together with a list giving the correspondence between node numbers and locations and times.

There may be less complicated ways of drawing the network than Figure 4 indicates. However, Figure 4 emphasizes the point that the only crucial properties of the network are the precedence

Figure 4 Network from space-time diagram



NODE NUMBER	1	2	3	4	5	6	7	8	9
LOCATION	D	A	B	D	C	B	C	A	B
TIME	t_8	t_6	t_4	t_1	t_3	t_5	t_9	t_2	t_7

relations. There are still three stops at location B, and the only possible movement of empty cars from location D to location C still requires a journey of four legs, the middle two corresponding to waiting periods at B.

principal properties

Three principal properties of the network portray the essential structure of any space-time diagram. First, if any set of nodes refers to the same location, then each node in this set is adjacent to some other node in this set; that is, there is a series of arrows (a path of arcs) that connect these nodes to each other without having to go through nodes outside of the set. This property is also true of sets of nodes corresponding to stops made by the same train; in fact, the events could be considered as being identified by train rather than location.

The second property relates to the first: at most, two arcs enter each node (arrows point in toward node) and two arcs leave each node (arrows point out). Furthermore, of the four arcs incident to any node, at most, two arcs (one directed inward, the other directed outward) refer to physical movement via a train, and similarly, two arcs refer to the purely temporal "movement" of waiting at a location. The fact that so few arcs are incident to any particular node, and that they can be so well categorized, is computationally advantageous.

The third property is the most important. If desired, the nodes could be numbered so that all nodes preceding any particular node have a lower number than that node, and all following nodes have a higher number. Mathematically, this property signifies a partial preordering of the nodes. In terms of the network, it signifies that it is impossible to trace through any portion of the network in the direction of the arrows and arrive at a place already visited. Because of this, the network is said to have no directed loops or cycles, i.e., to be *acyclic*. Thus, if there is a movement (flow) of empty cars through a portion of the network, it never returns to an earlier point, but instead always proceeds to a later point.

flow rules

The flow of cars over (or through) the arcs of the network must obey certain rules: (1) It must come from somewhere; (2) It must go somewhere; (3) It should be conserved; (4) It must be in the direction of the arrows. Obviously, the fourth rule must hold true, for otherwise empty car movements would appear to go backwards in time.

The first rule is also apparent. The cars must get into the system somehow: either a given car is available at the beginning, or else it is made available part of the way through. In the first case, for each location at which there are cars available, a number indicating the quantity can be associated with the first node for that location (each first node has no incoming temporal arcs). This quantity can then flow out from that node to other places in the network. Similarly, in the second case, if some cars are made available at a particular location part of the way through (say, because a customer releases some empty cars or because a shipment of empty cars arrives from another service area), they can be con-

sidered as available in the network at the first node for that location after they are released. This holds true because the cars cannot go until the time of the first train connection after their release; a node is defined in the network by such a connection. Thus, for every node at which cars have been made available from exogenous sources, a number indicating the net quantity of available cars can be associated with the node. These cars are then eligible to move from this node to other, later nodes in the network.

Similarly, the second rule holds true for cars to get out of the system: either a car is required at some location part of the way through, or else it is not used at the end of the time span. In the first case, empty cars, needed at a particular location due to customer requirements, can be considered as being required at the nearest node for that location preceding the time when they are actually required. This holds true because the cars must be at this location by the time of the last train connection and before the specified time (any later connection would miss this requirement); such a connection defines a node in the network. Thus, for every node at which cars are required, a number indicating the net quantity of required cars can be associated with the node. The required quantity should be moved to this node from other, earlier nodes of the network. In the second case, all cars made available but not required are left over at the end of the time span. These cars can be associated with the last node of each of the various locations, since they have not been required at earlier nodes (where each last node has no outgoing temporal arcs). The quantity is just the number of available cars not in use at the end: the total number of such cars can be determined, but the number at any one of the final nodes is an unknown quantity, since this depends upon how the cars were hauled around the network.

The third rule of flow is the principle of "conservation of flow": for any node, the amount *in* must equal the amount *out*. In simple terms, the rule states that at a given time the number of cars arriving at a location plus the number made available there must equal the number of cars leaving (including possible leftover cars, some of which might then go to a maintenance facility) plus the number required there.

Consider a specific fleet of empty cars, and suppose that these cars are homogeneous in nature, that is to say, any one car can be used instead of any other car. Suppose there is a system using these cars and that we have constructed a network from a space-time diagram describing their movement during a given time span. Suppose also that numbers have been assigned to the nodes of the network indicating the quantity of available or required cars at each node, so that rules of flow 1 and 2 are satisfied. Then any flow through the network that satisfies the rules of flow 3 and 4 corresponds to a possible movement through the system that satisfies the requirements of cars at the various locations at specified times. Usually, there are many such possible movements, and

the difficulty is not so much in constructing or obtaining such a movement, but in choosing between possible movements. This difficulty is the heart of the empty freight car allocation problem.

In the following section, the statement of the problem is formulated. An algorithm is then presented to solve this problem.

An algorithm for the allocation problem

Assume there is a network corresponding to a space-time diagram of the movement of empty cars. Let the nodes of the network be numbered consecutively from 1 to n for a total of n nodes, so that we can discuss node i or node j (where, of course, $1 \leq i \leq n$, $1 \leq j \leq n$). The arcs, then, can be referred to as *arc* (i, j). Each arc is either physical (a train connects two stations) or temporal (each waiting period at a station is between two consecutive train connections).

Assume further that the quantities of available and required cars are known, and let s_i be the number of cars available at node i. Let d_i be the number of cars required at node j. Without loss of generality, we can let $b_i = s_i - d_i$ for each node i. Then b_i is the net number of cars available at node i, and if $b_i < 0$, then this is to be interpreted as a demand of $-b_i$ cars at node i.

Now, let the unknown car movements (variables to be determined by the algorithm) be represented by x's, so we can refer to x_{ij} as the number of cars that flow from node i to node j over arc (i, j) assuming that arc (i, j) exists. There is one such variable for each arc in the network. The problem of constructing or obtaining car movements is to find a consistent set of values for the x's so that the flow rules are satisfied. However, something more is necessary to be able to choose between possible consistent movements.

Consider then, the assignment of a nonnegative cost c_{ij} to each physical arc (i, j). Let this cost be on a per unit hauled basis between node i and node j. It might be in dollars, and could represent the cost per car to haul a car over the particular leg of a route corresponding to arc (i, j). Then, we can speak of trying to find that set of values for the x's which minimizes the total cost, i.e., to find those car movements that satisfy the flow rules and give a total cost no greater than any other possible car movement. In terms of the dollar cost, we seek the least expensive car movements; in terms of distance cost, we seek car movements requiring the least total car-miles. This criterion has the following property: if it is not necessary to move a car from one station to another to fulfill requirements, this car is not moved. For simplicity, we can take c_{ij} on each temporal arc (i, j) to be zero, so that it costs nothing to store extra cars (nonzero storage costs require a more general algorithm than the one presented here).

problem statement

Now, we are able to write down formally the problem of empty freight car allocation. The problem is to find values for the variables x_{ij} which satisfy all of the following relations:

1. $\sum_i \sum_j c_{ij} x_{ij}$ is a minimum,
2. $\sum_j x_{ij} - \sum_k x_{ki} \leq b_i$ for each node i identified with the last stop at a station,
3. $\sum_j x_{ij} - \sum_k x_{ki} = b_i$ for each node i identified with a stop other than the last at a station, and
4. $x_{ij} \geq 0$ and integral for each arc (i, j).

These relations, of course, apply to the problem as defined, that is, any term with an x_{ij} in it only appears if arc (i, j) occurs in the underlying network. The first relation expresses the requirement that total cost should be minimized, where $c_{ij} \geq 0$ if (i, j) is a physical arc and $c_{ij} = 0$ if (i, j) is a temporal arc. The second and third relations express the principle of conservation of flow with the requirement that unassigned leftover cars must end up at the last stop at a station. For example, relation 3 states that, for every node except the last one, the net number of cars leaving a node (the number out *minus* the number in) must equal the net number of cars available at that node from external sources. Relation 2 states almost the same thing for terminal nodes, except the difference between the two sides of the relation represents the number of unused cars ending up at that node. Of course, when the sums in relations 2 and 3 are written out fully, there are, at most, two x's in each relation with a positive coefficient and two preceded by a minus sign. This follows from one of the basic properties of the underlying network mentioned earlier: there are at most four arcs incident to each node—two directed into the node, and two directed out. Outward arcs at a node are associated with a positive coefficient for the corresponding variable.

The fourth relation expresses the remaining rule: flow must proceed in the direction indicated by the arrow. Otherwise, we would have negative cars, a physical impossibility. There is the added requirement that x_{ij} be integral—answers given in terms of fractions of cars would be of little use.

a linear program

The problem as it has been formulated is a linear programming problem and can be solved using linear programming techniques.[6,7] However, this particular problem falls into a subclass of linear programs known as transshipment problems, and can be solved using general minimum cost-feasible flow network techniques.[6,8] The algorithm developed later is a special purpose cost-flow network algorithm designed to take advantage of the properties of the network that underlies this problem. However, considerable use is made of the fact that this problem is a network flow type, and the properties that hold true for network flow problems must also hold true here. In particular, there is one important property: if the supply data, the b_i's, are whole numbers, there is never a need to worry about fractional solutions, for answers are given in terms of integral numbers of cars.

method of solution

The algorithm uses an inductive method of solution. Instead of examining the whole network, we solve only a part of the net-

work at any one time (except at the end). Once a subnetwork is solved, we go on to another subnetwork. This second subnetwork is constructed by taking the first node and adding one more to it, and also including any arcs that connect this added node to the first subnetwork. The added node has all of its immediate predecessors already included in the subnetwork. The nodes immediately preceding a given node are at the tails of the arcs heading into the given node, so any flow that comes to the given node must come via its predecessors.

This progression from subnetwork to subnetwork can always be done by virtue of the acyclic property of the network. As mentioned earlier, the nodes could be numbered in such a fashion that all nodes preceding a given node have a lower number than that node, and all nodes following the given node have a higher number. We can thus partially preorder the nodes and proceed as follows: start with the node with the lowest number and consider that as the first subnetwork. To get the second subnetwork, add the second node plus any arcs that connect it to the earlier node. This process continues until all nodes have been added and the network is complete. It is not necessary to actually have the nodes numbered in this way as long as the rules of precedence are observed, but this serves to show that the inductive procedure is valid.

node designation

At any stage of the algorithm, the nodes that are in the subnetwork currently being examined are called *marked nodes*. The rest of the nodes are called *unmarked nodes*. Those unmarked nodes that are eligible to be added to the subnetwork to obtain the next subnetwork are *markable nodes*. Thus, even though a markable node is presently unmarked, all of its predecessor nodes must be marked. If, for example, the nodes are numbered in "order," then the unmarked node with the lowest number is always markable.

Given a subnetwork, then, we construct the next subnetwork by selecting any markable node, marking it (and including all arcs between this node and its predecessors), and considering this larger subnetwork of marked nodes and their arcs. For any subnetwork, the relations 1 through 4 must hold, just as if this were a complete problem in itself. The only difference in the relations proceeding from subnetwork to subnetwork is that adding a node might change a type 2 relation to a type 3 relation, and that another type 2 relation is added to the set of relations. The first change occurs if the added node replaces an already present node as the last node for a station, the replaced node becoming an intermediate node.

subnetwork properties

Thus, a sequence of subnetworks is created, each having an associated subproblem of the whole problem. Each subnetwork contains all earlier subnetworks and is contained in all later subnetworks. Similarly, the succeeding subproblems are progressively more restrictive, since an additional restriction on movement is added, and a restriction present in earlier subproblems may have been strengthened (from a type 2 to a type 3). If we

call a solution that satisfies all the restrictions 1 to 4 an *optimal solution*, a solution which satisfies 2 to 4 a *feasible solution*, and a solution for which at least one constraint of the relations 2 to 4 is not satisfied an *infeasible solution*, then it follows from the procedure that the following two properties hold:

- An optimal solution for the problem corresponding to some subnetwork is feasible for the problems associated with all earlier constructed subnetworks.
- If no feasible solution can be found for the problem corresponding to some subnetwork, then there are no feasible solutions for the problems associated with all later subnetworks.

These properties are used as follows: assume we have an optimal solution for a given subnetwork. A new network is then constructed as described previously. The current solution may or may not be feasible for the new problem— if it is, then it can be altered very simply to become an optimal solution to the new problem. If the current solution is infeasible in this new problem, it is because the added restriction, and only this restriction, is violated. The algorithm then attempts to satisfy the violated restriction in a sequence of steps, during this time maintaining the other relations at minimum cost. If the restriction cannot be satisfied, there is no solution to the problem, and we go no further. Otherwise, an optimal solution for this current problem is achieved, and we can continue to iterate. Iteration continues until either the problem is shown to be infeasible, or else an optimal solution to the problem as a whole has been obtained.

solution of specified subnetworks

Thus it remains for us to examine how the algorithm solves a specified subnetwork. Given a subnetwork, the algorithm *distinguishes* some of the arcs of the subnetwork. If we can go from one node of the subnetwork to another node of the subnetwork traveling only over distinguished arcs, irrespective of the direction of these arcs, then this set of distinguished arcs is characterized by only one such path. If the term "link" is used to refer to an arc without regard to its orientation (direction), then there are no loops in the subnetwork consisting solely of distinguished links. When all the undistinguished links are deleted from the subnetwork, what remains is called a *forest*. This forest may consist of many disjointed, separate pieces called *trees*. For each tree, every node is connected to every other node in the same tree by distinguished links with a unique connecting path. Thus, if there are k nodes in a particular tree, there are $k-1$ links in the tree. In each tree, one particular node is chosen to be the *source* or *root* of that tree. Note that a tree need not have any links; it could just consist of a single node, its root. It is also true, by the definition of the forest and trees, that every node belongs to some tree. The forest itself might consist of a single tree, or there might be as many trees as there are nodes in the subproblem.

Many trees are possible in any one subproblem, that is, there are many possible ways to select the distinguished links. As the

algorithm progresses, the forest may be reformed many times for any given subproblem according to explicit rules. However, there is one general principle: the only nodes allowed to be considered as roots are those nodes that correspond to the last node at a station in the given subnetwork. Since each tree must have exactly one root, only as many trees can be in the forest as there are stations. Note that the only nodes allowed to be roots are those nodes allowed to have extra or leftover cars.

The forest with its trees is important for the algorithm, because flow is only permitted over distinguished links. Furthermore, at any stage in the solution, each node (of the current subnetwork) having extra cars is a source node for some tree. In linear programming terms, a "slack" variable is associated with each source node (the slack variable represents the number of empty cars at that node). The slack variables together with the variables of the distinguished arcs constitute a "basis" for the problem connected with the subnetwork under consideration.

potentials

Associated with each node i in the subnetwork is a dual variable or potential Π_i, which, as discussed later, represents the marginal cost of getting an additional car to node i from the source in the tree of node i, assuming there is a car that can be sent. The potential for the source node of each tree is defined to be equal to zero. For each tree, the potentials of all other nodes in the tree can be calculated by using the following rule:

> If node i and node j both belong to the same tree, and arc (i, j), which has cost c_{ij}, is distinguished, then, if either Π_i or Π_j is already determined, the other potential can be determined by using the relation $\Pi_i + c_{ij} = \Pi_j$.

The fact that potentials can always be calculated in such a fashion results from the definition of trees. Since the potential of each source node is zero and the costs are all nonnegative (with the costs on the temporal or waiting arcs identically zero), by using the relation just established, we can show that the potential of every node must be nonnegative for a minimum cost solution.

additional restriction

Previously we saw that there are many ways to form the trees for any given subproblem, and that the algorithm only allows certain nodes, those corresponding to the last node at a station, to be sources. Now, another restriction is added: for any two nodes i and j of the network that are connected by an arc (i, j), the potentials must satisfy $\Pi_i + c_{ij} \geq \Pi_j$.

It is not obvious that the inequality is satisfied. However, such inequalities together with the zero restrictions on the source potentials constitute the restrictions of the "dual" linear program, which must be satisfied by any optimal solution to the problem. During the application of the algorithm, arcs are distinguished (and are made undistinguished) in such a fashion that these restrictions are always satisfied for the subproblem at hand (in linear programming terms, the solutions are dual-feasible).

The potentials have a very real interpretation as marginal costs. For any node i, Π_i represents the cost of getting one additional car to node i by reducing the number of available cars by one at the source in the tree of node i, and adjusting the flow of cars over the distinguished arcs between node i and its source. If all distinguished arcs in this path are oriented from the source toward node i, getting one car from the source to node i amounts to sending one car at a cost Π_i. However, the adjustment of flow may occur in a negative way by not sending a car that was assigned earlier to a particular arc. For example, suppose some cars (flow) are going to be sent from node i to its neighboring node j over the distinguished arc (i, j). One additional car is obtained at i and the number of cars available at j is reduced by one merely by holding at i a car that was supposed to go from i to j. The situation can be thought of as sending from node j to node i a flow of one car, but by traversing the arc in the wrong direction, this car "cancels out" a car sent in the correct direction. Needless to say, this only makes sense if there is at least one car that can be canceled on each arc concerned. But, if it is assumed that there are such cars, then the cost of obtaining one additional car at node i by "sending" it from the source in the tree of node i is just Π_i. This cost is obtained by totaling the costs along the path from the source to node i, adding a cost for each arc over which the flow has been increased by one unit, and subtracting a cost for each arc over which the flow has been reduced. In fact, the potentials were defined recursively in this way.

extra car cost

It was noted that the potential of every node must be nonnegative. In other words, using the cost structure assumed previously, sending additional cars to any node is always a nonnegative cost. Thus, to get an extra car to a node, it costs at least as much as not sending it at all. Hence, if all requirements were satisfied by using the current availabilities, there would be no incentive to send extra cars to any station, unless there were a less costly way of sending them.

The latter situation, however, does not occur. Suppose, for example, that node i and node j were connected by an undistinguished arc (i, j). The marginal cost of supplying cars to node i is Π_i and to node j is Π_j. Consider the possibility of supplying these cars to j by sending them to i and then to j via arc (i, j) rather than to j directly via distinguished links. Here, the cost per car of supplying them is just $\Pi_i + c_{ij}$, which is the cost of sending them to i and traversing arc (i, j). Now, if this cost were less than Π_j (the cost of sending cars directly to j) there would be an incentive to reroute some cars. But this can never be the case, since the restriction was posed that, for all arcs (i, j), the potentials must satisfy

$$\Pi_i + c_{ij} \geq \Pi_j$$

We can easily extend this reasoning to conclude that it always costs at least as much to send a car as not to send it, and, if the

car must be sent, one way of doing it at least cost is to send the car to the node from the source in the node's tree via the distinguished links of the tree.

algorithm operation

Thus, the algorithm works in the following fashion. Assume that we have solved the problem with a given number of marked nodes and are about to perform the inductive step. We have an optimal solution for this last problem, with flows, potentials, trees, and sources, with flows from sources to nodes only occurring on arcs of the trees, and with a marginal cost of each such flow given by the potential of the node. Now, suppose that the new node added to the problem has a requirement associated with it, such that the current solution is infeasible in the new problem. Thus, it is necessary to send flow to this added node from a neighboring tree (a tree which, with the distinguishing of one more arc, would connect to the added node). The arc giving the least increase in marginal cost is chosen to be the arc over which flow will be sent. It is distinguished, and as much flow as possible is sent, up to the required amount. If the required amount can be sent, the optimal solution has been obtained. If not, either there was not enough flow at the source node, or else some link restricted the flow (which can happen if the actual flow in an arc is reduced to zero because it was canceled by flow in the opposite direction). In the latter case, make the restricting link undistinguished. In both cases, the node with the requirement is connected by a set of distinguished arcs to a tree that, in essence, has no source. The tree itself must then be connected to some neighboring tree to supply more flow to the given node. The link that makes this connection is chosen from the set of possible connecting links by the least cost rule, and the procedure is iterated.

This procedure will terminate either with the filling of the requirement, or else by showing that the requirement cannot be filled and, hence, that this subproblem as well as the problem as a whole is infeasible. That the algorithm does work, and does so in a finite number of steps, can be shown from the basic principles previously described when some rules for resolving choices are included. The detailed statement of the algorithm that follows contains the complete rules by which the algorithm can be proved. However, the proof follows trivially from one presented by Ford and Fulkerson[8] if we note that this algorithm is just the "out-of-kilter" algorithm when a particular ordering of the nodes is used, when all costs are nonnegative and some are identically zero, and when solutions are basic and are kept that way (see, for example, Johnson[9] on this last point).

computation of algorithm

The algorithm as now presented is for hand computation, to aid in an understanding of the basic steps. For machine computation, many details can be incorporated to increase efficiency. They are not included at this point to avoid unnecessary complexity. However, some of these details are discussed later. The terminology used in the statement of the algorithm has already been introduced. As is readily seen, each time Step 1 is performed, an inductive step

is taken, and Step 3 is the routine that fills the requirements at least cost. The algorithm is as follows:

Step 0: Initially, assume that all nodes without entering arcs are markable. All nodes are presently unmarked.

Step 1: Let n_0 be some unmarked markable node. If no such node exists, stop here; an optimal solution to the whole problem has been found. Otherwise, set $\Pi_{n_0} = 0$ and mark node n_0. If node n_0 is the first stop at some station, then go to Step 2. Otherwise, let n_1 be the node that was formerly the last stop at a station and has now been replaced by node n_0. If node n_1 is not a source, then go to Step 2. Otherwise distinguish arc (n_1, n_0) and carry the extra flow at node n_1 forward over this arc to node n_0, regard n_1 as no longer a source, and continue.

Step 2: If there is no requirement to be filled at node n_0, consider node n_0 to be a source and return to Step 1. Otherwise, continue.

Step 3: Let node n_2 be in the same tree as node n_0, and let node n_3 be marked, but not in n_0's tree. For each such pair (n_3, n_2) such that arc (n_3, n_2) exists, find the increase in marginal cost, $V(n_3, n_2)$, by the formula

$$V(n_3, n_2) = \Pi_{n_3} + c_{n_3 n_2} - \Pi_{n_2}$$

Let V equal the smallest $V(i, j)$ obtained, so that V is the minimal increase in marginal cost. If V cannot be found because no $V(i, j)$ exists (i.e., no such arc (n_3, n_2) exists in the network), then stop here; the problem is infeasible. Otherwise, increase Π_i for each node i in n_0's tree by the value V. Let the arc which gave the value V be arc (n_3, n_2) (if more than one arc gives this value, choose arbitrarily among them), and distinguish this arc. The tree including node n_0 has now been joined to another tree having a source. Now, send flow from the source in this enlarged tree over distinguished arcs to node n_0 until either

(a) the requirement at node n_0 is filled, or
(b) the requirement at node n_0 is not filled, but no more flow can be sent.

In condition (a), return to Step 1. Otherwise, condition (b) happens if either (1) the flow over some arc in the path from the source to node n_0 has been reduced to zero (by "canceling"), or (2) Case 1 did not occur, but the extra flow at the source in the tree has been exhausted (the requirement at node n_0 was larger than the surplus at the source).

In Case 2, do not regard the source as a source any longer and return to Step 3. Otherwise, take the distinguished arc on which flow has been reduced to zero (if there is more than one such arc, choose arbitrarily among them), and make this arc undistinguished. The tree that includes node n_0 has now been broken into two trees, with n_0 in one tree and the source in the other. Return to the beginning of Step 3.

Figure 5 Sample problem network

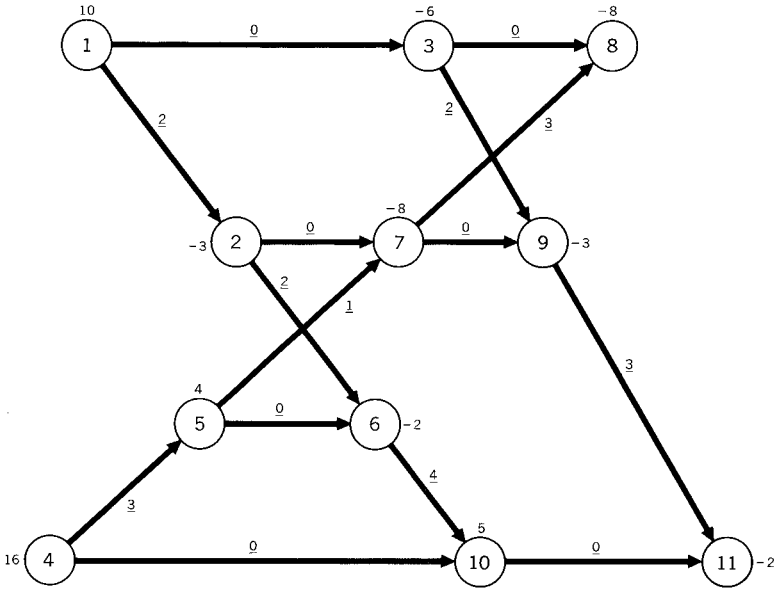
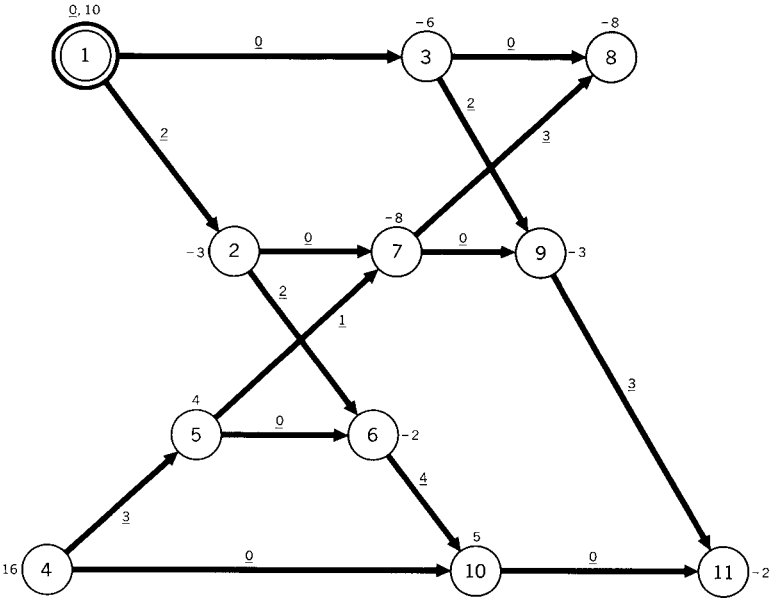


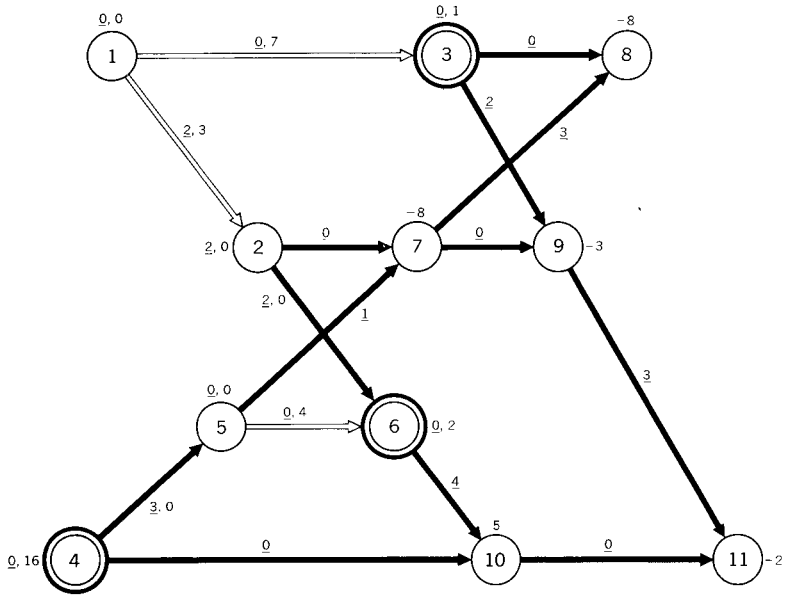
Figure 6 Network after cycle 1



example

To illustrate this algorithm, a small eleven-node sample problem is used. This problem has three trains and four stations. For expository purposes, we adopt the convention that all nodes drawn on the same horizontal line refer to the same station. The nodes are numbered from one to eleven, the number for each node appearing within the node on the diagram. The availabilities at each node are given by the number appearing next to the node

Figure 7 Network after cycle 6



(where a negative number signifies a requirement for that many cars). The cost associated with each arc of the network appears as an underscored number next to the arc. The sample problem is shown in Figure 5.

During the course of the algorithm, potentials are assigned to the nodes. These potentials appear as underscored numbers, one next to each node. Since each marked node has a potential, unmarked nodes can be recognized as having no underscored number next to them. In addition, the amount of flow sent over any given arc appears as a number, not underscored, next to the arc. Distinguished arcs are represented by double lines as opposed to single lines, and source nodes have an extra circle around the node number. We progress with the sample problem as follows:

Cycle 0: Initially, nodes 1 and 4 are markable.

Cycle 1: Of the markable nodes, mark node 1 first by setting its potential equal to zero. Going directly to Step 2, mark node 1 as a source (since it has a surplus of ten cars) and return to Step 1. This cycle has been completed, and the network appears as in Figure 6.

Cycles 2–6: We can readily ascertain that if nodes 2 through 6 are successively chosen to be n_0, the steps of the algorithm give us three different trees, the first with nodes 1, 2, and 3, the second with nodes 5 and 6, and the third with node 4. The source for each tree is node 3, 6, and 4, respectively. Thus, at the end of cycle 6, the status of the network appears as in Figure 7.

Cycle 7: From the markable nodes 7 and 10, choose node 7 to be n_0. Here, it takes three attempts to fill the demand of node 7.

Figure 8 Network after cycle 7

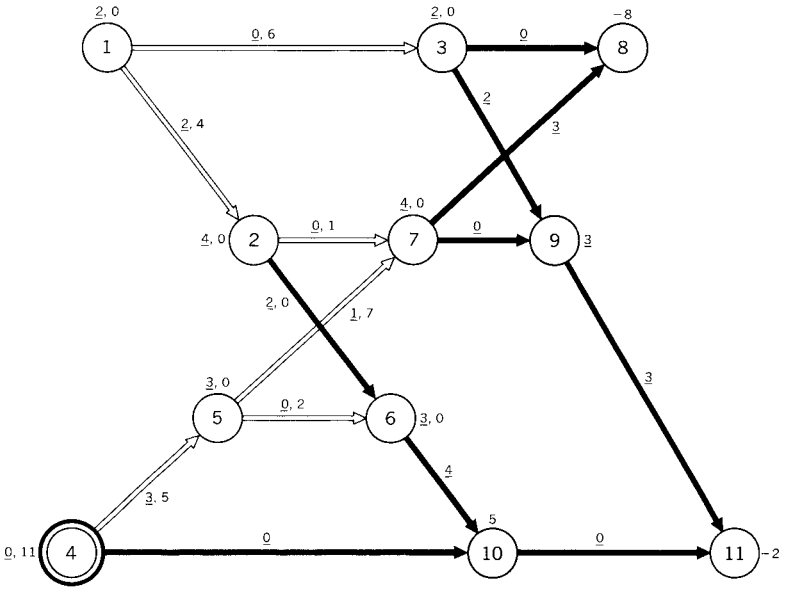
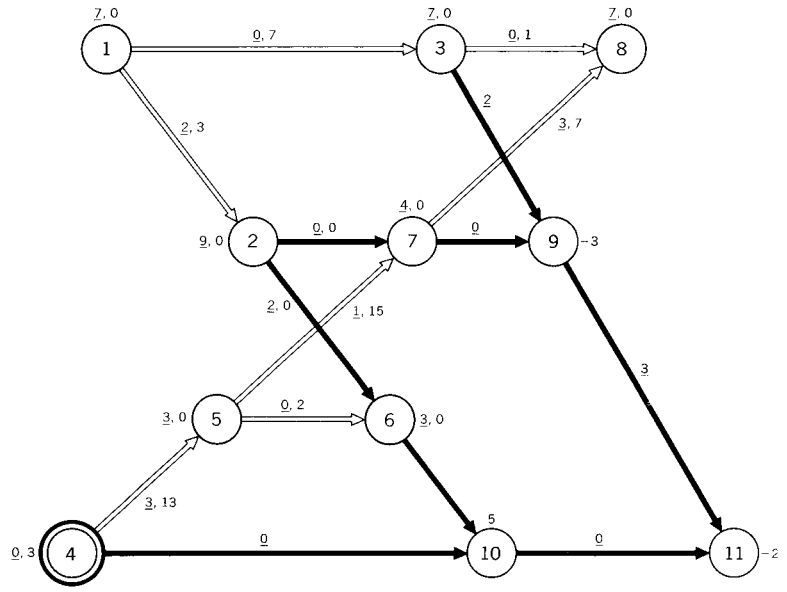
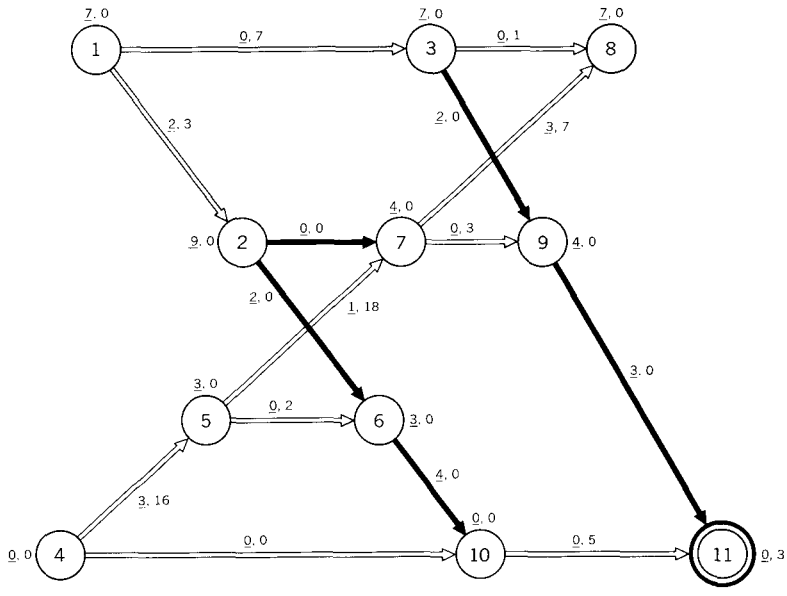


Figure 9 Solution to eight-node subproblem



The first flow comes from node 6 via arc (5, 7). When node 6 is exhausted, the next least cost flow is from node 3 via arc (2, 7). Finally, flow must be sent from node 4 to satisfy the demand at node 7. As shown in Figure 8, the final network of cycle 7 consists of a single tree with node 4 as its source, and with potentials and flows as indicated.

Figure 10 Optimal solution to sample problem



Cycle 8: Of the markable nodes 8, 9, and 10, choose node 8 to be n_0. It takes two attempts to fill the demand of node 8. The least cost route is via arc (3, 8). However, sending one unit of flow along this route reduces the flow over arc (2, 7) to zero, so that the tree separates. The remaining demand can then be satisfied at least cost by using arc (7, 8), and a single tree results, still with node 4 as its source. Cycle 8 is therefore terminated, and the optimal solution to the eight-node subproblem is given in Figure 9.

Cycles 9, 10, 11: Proceeding, choose nodes 9, 10, 11 successively to be node n_0. At the end of cycle 11, all nodes have been marked, and the algorithm terminates with an optimal solution to the sample problem. This solution, pictured in Figure 10, consists of a single tree with its source at node 11. The car movements necessary to satisfy the original requirements from what is available can be read from the figure. For example, the train that goes from node 4 to 5 to 7 to 8 picks up sixteen cars at node 4, picks up two more at node 5, sets off eleven cars at node 7 and hauls the remaining seven cars to node 8.

Implementation of the algorithm

The algorithm has been incorporated into a computer program,[10] which is written in PL/I. Input is in terms of stations, train schedules, costs, and requirements and availabilities. Output is in terms of the car movements at the stations and on the various trains necessary to achieve the optimal allocation of empty cars. The input and output were chosen to be in this form so that

railroad operating personnel might be closer to the context in which the program is used.

The program is organized into two main parts. The first part is basically a data processing procedure to match and cross-reference input information. The second part, described later, performs the steps of the algorithm and presents the results as output.

The first part internally creates the network to which the algorithm is applied (in essentially the same way as was described earlier): each train stop is taken to be a node and the times of the stops given by the train schedules are used at each station to create the precedence relations between the nodes at the station. Precedence relations between nodes for each train are, of course, automatically given by the order of the stops made by the train. Costs are assigned directly to the arcs, where the identically zero costs of storing empty cars (assumed here) are implicitly, rather than explicitly, kept. The allocation of computer memory capacity is dynamic, and considerable use is made of the capability of PL/I in this regard, including the use of push-down stacks.

The only involved calculation in this first portion of the program deals with the computation of requirements and availabilities at nodes, since these may actually occur between nodes and, as explained in the section on patterns of movement, must be converted to occur at the nodes. It is done in a straightforward manner, essentially by "netting out" the requirements (and/or availabilities) that occur between each pair of adjacent nodes at each station.

Within the algorithm itself, considerable advantage is taken of the special properties of the underlying network. The induction of the algorithm is, of course, founded on the acyclic nature of the network. However, the fact that, at most, four (and at least two) arcs are incident to each node is also invaluable, for this means that data can be kept in a node-oriented fashion, i.e., for each node a list of four numbers giving the adjacent nodes can be kept, and if there are not four adjacent nodes, the corresponding numbers in the list can be given a value of zero, since the position in the list determines whether the node precedes or follows the other nodes at the same station or on the same train.

bit assignment

Similarly, a set of four bits can be kept for each node to tell whether the corresponding arcs are distinguished or not. Data may also be kept as to the supply at each node, the potential at each node, the amount of flow leaving to go to the next station node (the amounts of flow entering the node are given by the amounts leaving the preceding nodes), and the explicit cost of leaving the node (by waiting at the station the cost of leaving the node is implicitly zero, and the costs of entering the node are given by the costs of leaving the preceding nodes). Since it is necessary to know whether each node is marked or not, one bit is used for this purpose, and another bit can be used to determine whether a given node is in a tree that is currently under investigation or is in some other tree.

During the course of the algorithm, flow must be sent out from the sources of the trees. Thus, if the number of the node in the same tree, which is one step closer to the source of that tree than the given node, is kept at each node, the path that the flow must take to get from the source node to some node in the same tree can be found by tracing backward from the node to the source. (This tracing procedure must actually be performed twice: once to evaluate how much flow can be sent, and again to change the level of flow to the new value.) Note, however, that since four nodes, at most, are adjacent to each given node, a two-bit pointer at each node is sufficient to indicate which of the adjacent nodes is the next node closer to the source.

Hence, a total of eight bits can be kept for each node, and the bit-handling capabilities of PL/I are used advantageously in the program. The use of these bits within the program creates some bookkeeping difficulties within Step 3 of the algorithm, but these can be treated directly. For example, since trees are joined and sometimes separated, a routine must be added to update the tree status bits of the involved nodes whenever this happens. Furthermore, the source direction bits may also be changed on these occasions, and a routine must be used to accomplish this purpose. Both of these routines are straightforward: in the present program, the tree status routine essentially uses a push-down stack, and the other routine uses a backward tracing method from node n_0 to the node at which n_0's tree was joined or separated.

program use

Although the program can be used for long-range planning of expected car movements, it seems likely that the main use is in planning short-range car movements in a multiperiod context. Other approaches[11,12] may be of greater use in long-range car fleet utilization. The program might be run once per time period, where the length of the time period depends on the frequency of car movements and on the accuracy of the predicted volume. The time span represented by the problem could equal or slightly exceed the number of time periods needed to haul a single car across the system under consideration. The program, then, could be run once every time period for all trains that have at least part of their schedule within the time span. However, only the results for the first time period during the span are actually used, since the next time the program is run, more accurate information is available regarding the requirements and availabilities of the later time periods. This approach has the effect of "smoothing" the discontinuity between time periods because some cars, though not required within the first time period, may be repositioned in anticipation of what the later time periods may require.

Extensions to the algorithm

Although developed for railroad applications, the space-time diagram and the corresponding network are common to many areas and modes of transportation. For some industries, only

the terminology need be changed to make everything mentioned apply. Empty barge-line operations are a good example. In some cases, minor modifications of the algorithm, or reinterpretation of the formulation, can also result in wider applications.

The algorithm is operational in its present form, but many extensions can be made at all levels of detail. One small modification is the addition of upper and lower bounds on the number of cars hauled over each leg of the train schedules. The inclusion of these in the algorithm is straightforward (in fact, bounds are included in the program[10]) and the necessary changes can be deduced from the appropriate specialization of the "out-of-kilter" method given in Ford and Fulkerson.[8] Putting bounds on the number of cars at a station, however, is a more difficult problem, and requires a more general algorithm.

A routine can also be built into the algorithm for handling shortages that may occur. It might be as simple as deferring a shortage to a later time (the program does have this capability:[10] there is an option that, if the situation is infeasible at a given node, as much as possible is supplied to that node, and the remaining shortages are transferred to the next node at the same station, so that the requirement might be satisfied as soon as possible). Or a more sophisticated method of choosing between shortages at various locations can be devised, should they occur. In this way, priority requirements can be implemented by solving the problem once for each priority, updating the requirements each time.

The algorithm is designed to handle a single homogeneous class of car; however, there are times when certain cars are partially interchangeable, for example, when a car of a certain class can be "regraded" to haul, say, two different types of products. "Regrading" of cars can be incorporated in the model by replicating the network once for each car grade. These networks are then connected together by arcs that represent downgrading and upgrading, i.e., the arcs connect nodes of different grades at the same location. Upgrading arcs need only be present at locations at which there are upgrading facilities. In this fashion, high-grade type cars can be converted into lower grade cars, and low-grade type cars can be converted into higher grade cars by routing them through the upgrading facility.

The use of the algorithm and the program can be extended by tying it in to various systems capabilities, such as teleprocessing, where some data could be automatically entered using detection devices, and other data could be entered from railroad yard or customer locations. Similarly, output could be sent back to the concerned parties. By decreasing the number of surplus cars existing on most railroads today and increasing car utilization, such a system earns its own way by releasing capital investment for other purposes.

CITED REFERENCES

1. A. Orden, "The transshipment problem," *Management Science* **2**, No. 3, 276–285 (April 1956).
2. G. Feeney, *Controlling the Distribution of Empty Freight Cars*, Tenth National Meeting, Operations Research Society of America (1957).
3. C. D. Leddon and E. Wrathall, *Scheduling Empty Freight Car Fleets on the Louisville and Nashville Railroad*, Second International Symposium on the Use of Cybernetics on the Railways, Montreal, Canada (October 1–6, 1967).
4. E. Wrathall, "Empty freight car scheduling and extensions," *Transportation: A Service*, J. S. Coutinho, (Editor) New York Academy of Science, New York, New York (1968).
5. *Scheduling Empty Freight Cars on the Louisville and Nashville Railroad*, Application Manual E20-0283, International Business Machines Corporation, Data Processing Division, White Plains, New York.
6. G. B. Dantzig, *Linear Programming and Extensions*, Princeton University Press, Princeton, New Jersey (1963).
7. *An Introduction to Linear Programming*, Application Manual E20-8171, International Business Machines Corporation, Data Processing Division, White Plains, New York.
8. L. R. Ford and D. R. Fulkerson, *Flows In Networks*, Princeton University Press, Princeton, New Jersey (1962).
9. E. Johnson, "Networks and basic solutions," *Operations Research* **14**, No. 4, 619–623 (July–August 1966).
10. W. W. White, *A Program for Empty Freight Car Allocation*, IBM Contributed Program Library, 360D.29.4.002, International Business Machines Corporation, Program Information Department, Hawthorne, New York (January 1968).
11. B. Avi-Itzhak, B. A. Benn, and B. A. Powell, "Car pool systems in railroad transportation: mathematical models", *Management Science* **13**, No. 9, 694–711 (May 1967).
12. A. R. D. Norman and M. J. Dowling, *Railroad Car Inventory: Empty Woodrack Cars on the Louisville and Nashville*, IBM New York Scientific Center Technical Report 320-2926, New York, New York (January 1968).