Proceedings of the 2002 Winter Simulation Conference
E. Yücesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes, eds.

A RECURSIVE METHOD FOR TRAFFIC MANAGEMENT THROUGH A COMPLEX PATH NETWORK

Michael Norman

Brooks-PRI Automation - Planning and Logistics Solutions
5245 Yeager Road
Salt Lake City, UT 84116, U.S.A.

ABSTRACT

Many simulation models contain one or more transport systems where some type of vehicle, perhaps an AGV, a fork truck, a shuttle, or a human being, travel along predefined paths. Locations along the paths may specify where loads carried by the vehicle transfer to or from it, where some activity takes place requiring the vehicle's presence, or where vehicle routing logic executes. This paper focuses on the last topic, the routing of vehicles along a path system where alternate paths exist and path selection is determined at so called "routing nodes" based on a dynamic analysis of traffic congestion along each possible route towards some destination. Other routing nodes further along each path present more combinations of possible interim paths. A recursive search algorithm is presented to iteratively evaluate each possible route when a vehicle encounters a routing node. The vehicle is directed along a path with the least overall congestion towards its destination. A sample model demonstrating this algorithm implemented in the AutoMod software is used for illustration. Other simulation products may have the features to support this type of vehicle routing control algorithm.

1 AUTOMOD'S PATH MOVER SYSTEM

The AutoMod simulation software contains a material handling system called the Path Mover, in which vehicles move along guidepath, carrying loads from pickup locations to delivery locations (called "control points"). Path Mover vehicles can represent manually operated lift trucks, people, or computer-controlled vehicles (any type of movement system in which vehicles follow a specific path or route).

By default, AutoMod vehicles in a Path Mover system use the shortest route between load pickup and delivery points. Where intersections offering more than one path segment to another control point exist, the default routing can be over-ridden by using a built-in "location selection function". This function defines the next control point a vehicle claims in its route, or if the next path segment is blocked, the transfer (a construct automatically inserted by AutoMod at path intersections) may offer an alternate path.

1.1 Location Selection Function

Path Mover vehicles automatically call the location selection function each time they have a choice of determining the next control point to claim on the way to their destination. The function provides the opportunity to select from a list of subsequent control points to the current location and returns the control point to which the vehicle will travel next (returning null from the function will direct the vehicle to take the shortest route to its destination, the default). The location selection function is a convenient way to make local decisions for vehicle routing.

```
begin pm location selection function
    choose a location from among theLocList
    whose remaining space is maximum
    save choice as Vchoiceloc
    return Vchoiceloc
end
```

The Path Mover code example above uses the location selection function to route a vehicle via the next control point with the most remaining space; control points have a capacity, one unit of which is claimed for each vehicle attempting to travel to or by it, at which time that claim is released.

1.2 Alternate Timeout

Path Mover vehicles traveling to their destinations via the shortest path will be delayed any time that path is blocked. Path Mover systems provide default settings for alternate path timeout time and alternate type when a possible path exists. These parameters may be edited as attributes of individual transfers between sections of path.

The *Alternate Time Out* is the length of time a vehicle attempts to take the default path at an intersection before taking an alternate route; the system default is 60 seconds, which may be changed.

When path is drawn, the AutoMod software creates transfers between intersecting sections of path. The *Alternate Type* for these transfers is a rule inherited from the default, but may be individually edited after AutoMod creates them. The rule choices for *Alternate Type* are: None, Round Robin and Branch, with None being the system default (which defeats alternate routing altogether).

If Round Robin has been selected as the alternate routing for a transfer at a path intersection, vehicles attempt to take each intersecting path in the order it was defined, until a successful transfer is made. If Branch routing is invoked, vehicles attempt to take the path selected in the transfer's Branch select list. If that path is not available, vehicles take the alternate path selected in the alternate select list (both select lists present the user with a choice from the set of each of the possible paths available from that transfer).

The use of "standard" alternate routing techniques offers the advantages of simple editing within AutoMod's user interface. If the system path being modeled has single intersections where alternate path routing takes place, and that decision is based on Round Robin or Branch methods, the standard features of alternate path selection are the most straightforward and should be used. If the next path segment chosen is based on some other local decision criteria, the location selection function may provide a good method. A more flexible method may be needed in more complex situations.

2 SEMICONDUCTOR FAB AMHS PROBLEM

The overhead monorail automated material handling system (AMHS) in some semiconductor fabrication facilities is an example of the type of system that might use a more complex routing mechanism to avoid vehicle congestion and thereby decrease product wait-for-pickup and delivery times.

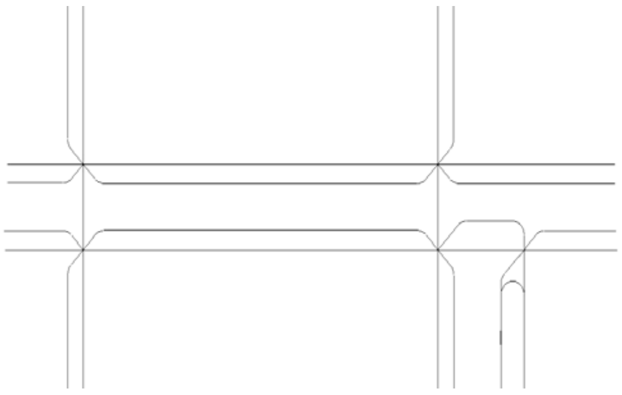

Figure 1: AMHS Track with Turntable Intersections

Some AMHS tracks include turntables, which act as "routing nodes" offering a selection of different possible track segments as vehicles travel between various locations. Vehicles moving along the unidirectional track may be redirected at a turntable from the most direct path (shortest physical track distance) to some other track segment where less congestion lies ahead. Some systems provide the use of a flag at each routing node to optionally use this feature. The remaining path may also be re-evaluated at each new routing node along the way to the vehicle's final destination to account for changes in congestion that may have occurred during travel.

3 ROUTING ALGORITHM

A "look ahead" evaluation mechanism for routing of vehicles requires a more complex modeling solution. The AutoMod software may be used to reroute vehicles to new interim locations while traveling to a final destination. This section presents an algorithm for changing the default route of a vehicle based on downstream congestion on each possible path segment towards a vehicle's final destination.

Upon approaching or arriving at a routing node, each vehicle evaluates a set of possible paths towards a final destination. This evaluation is done by combining path segments towards the destination and comparing the "weighted value" of each potential route. Each path segment *weight* is determined by the type of node next encountered (for simplicity, in this example all nodes are assumed to be the same type with equal weight) and the number of vehicle claims currently at that node. Path construction for evaluation is done through a recursive branch-and-bound type search.

The following subsections describe the decision process for the routing algorithm.

3.1 Vehicle Approaching Routing Node

If a vehicle is approaching a node assigned by previous alternate routing, that interim destination is cleared and the vehicle is reassigned to its final destination. If this node is also a routing node, the route analysis function will be called again.

- cancel previous assignment of this node as a temporary destination (from prior routing)
- set vehicle destination to final destination

Original call to Route Analysis Function:

- clear best weight, best path variables
- function passes current location, final destination, path constructed (initially null), path weight (initially 0)

3.2 Route Analysis Function

- increment the recursion level (in case the search depth is being limited)

Check if this branch (path) of the analysis is complete:

- the location passed is the final destination
- the recursion level has reached a preset depth (the number of recursions may be set to an arbitrary value to limit the computation time and the number of path segments ahead to evaluate)

If so:

- if the path distance to the final destination from the location under consideration is less than the total path distance from the current location to the destination then recalculate the weight of the path
  - if the new weight is less than the previously saved best weight then save this weight as the new best weight
- decrement the recursion level
- remove the last location from the current path construction and return

If not:

- determine next adjacent control point to the last point evaluated
  - if it is the current location of the vehicle or the node from which this search started, then this search has looped around
    - ➢ decrement the recursion level
    - ➢ remove this location from the path under evaluation
    - ➢ return
  - otherwise insert this location into the path for evaluation
  - call the route analysis function (recurse)
- if all adjacent control points to the last point being evaluated in the current path have been checked
  - decrement the recursion level
  - remove the last location from the path under evaluation
  - return

3.3 Search Pattern Example

Assume a vehicle approaches a routing node where four alternate paths may be taken (lanes 1 – 4), followed by an intersection of the four paths. This four-lane group is followed by a second (lanes 5 – 8) and third (lanes 9 – 12) four-lane set of alternate paths.

The pattern of lane (#) evaluation would be:

1.	current location (0)	(recursion level 1)
2.	0 + 1	(recursion level 2)
3.	0 + 1 + 5	(recursion level 3)
4.	0 + 1 + 5 + 9	(recursion level 4)
5.	0 + 1 + 5 + 10	
6.	0 + 1 + 5 + 11	
7.	0 + 1 + 5 + 12	
8.	0 + 1 + 6	
9.	0 + 1 + 6 + 9	
10.	0 + 1 + 6 + 10	
11.	0 + 1 + 6 + 11	
12.	0 + 1 + 6 + 12	
13.	0 + 1 + 7	
14.	0 + 1 + 7 + 9	
15.	0 + 1 + 7 + 10	
16.	0 + 1 + 7 + 11	
17.	0 + 1 + 7 + 12	
18.	0 + 1 + 8	
19.	0 + 1 + 8 + 9	
20.	0 + 1 + 8 + 10	
21.	0 + 1 + 8 + 11	
22.	0 + 1 + 8 + 12	
23.	0 + 2 + 5	
24.	0 + 2 + 5 + 9	
25.	0 + 2 + 5 + 10	
26.	0 + 2 + 5 + 11	
27.	0 + 2 + 5 + 12	

And so on… in general, the number of full path evaluations will be the product of the number of alternate paths at each level. The route analysis function is called somewhat more due to calls made during the construction of each path. Each evaluation could also reach different levels depending on whether the path was not getting closer to the final destination or had looped back on itself.

4 AUTOMOD IMPLEMENTATION

An AutoMod model has been created to demonstrate the routing algorithm. The path layout (Figure 2) shows how

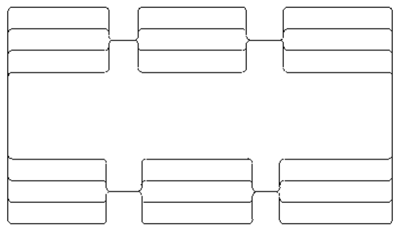

Figure 2: Example Model Path

the routing alternatives described in the previous section could be implemented in two areas of a closed loop. Before entering each of the three sets of four lane groups, vehicles choose a three-lane path through the entire twelve-lane section of the path network (note that addition of other routing nodes between each set of four lanes would provide the opportunity to re-evaluate conditions and change the path through the rest of that section).

4.1 Path Mover Vehicle Control

This routing control algorithm can be implemented in the AutoMod software with 2 functions and about 50 lines of AutoMod code. The first function is called "decelerate ok" and is automatically invoked by each vehicle in a Path Mover system every time the vehicle approaches a control point. It is typically used to determine whether the vehicle must stop at that location and if so, needs to begin decelerating. Since the vehicles do not stop in this example except to pick up or drop off a load, this function provides an opportunity to make a routing decision without forcing the vehicle to stop at a routing node (note code comments below are encapsulated between "/*" and "*/").

4.2 *decelerate ok* Function

```
begin pm decelerate ok function
/* approaching loc set by route function? */
    if stopLoc = theVehicle Anextloc and
        theVehicle Asavejob <> null then begin
        /* set back to final destination */
        set Vtempjob to
            theVehicle current schedjob
        set theVehicle current schedjob to
            theVehicle Asavejob
        cancel Vtempjob
        /* cancel routing move - don't stop */
    end

    print stopLoc to Vs
    /* approaching routing decision point? */
    if Vs length > Vs index("pick") then begin
        set Vweight to 999999
        set Vpath to null
        set Vorigin to stopLoc
        set Vdest to theVehicle destination
        set VTotDist to
            Vorigin path distance to Vdest
        /* set a new temporary route */
        call route(stopLoc,null,0)
        set theVehicle Apath to Vpath
    end
    if theVehicle Apath size > 0 then begin
        set theVehicle Anextloc to
            theVehicle Apath first
        remove first object from
            theVehicle Apath
        set theVehicle Asavejob to
            theVehicle current schedjob
/* route via chosen best next loc */
        dispatch theVehicle to
            theVehicle Anextloc
        set theVehicle current schedjob to
            theVehicle schedjobs last
    end
    return false   /* default - do not stop */
end
```

Each vehicle carries an attribute containing a list of the locations for the path chosen. As the vehicle navigates through the temporary path assigned and each location is encountered, that location is removed from the list and the next interim location on the list is assigned. Once the route has been finished or a new routing node is encountered, normal control is returned to the vehicle to make its way 'automatically' to its destination or a new temporary route is created and applied.

4.3 *route* Function

The *route* function is initially called from the decelerate ok function, and then recursively calls itself as potential paths are constructed and analyzed. The control point claims along each path are used to determine a "weight" to be assigned for that path. In this simple example, the least weight is assumed to be the least congested since fewer vehicles are currently assigned to pass those points. More complex route evaluations might include factors for different types of stations. For example, these factors could reflect typical delay times associated with those locations. Historical data could be used to tend to migrate vehicles away from longer path times.

Function parameters in AutoMod act as local variables to that instance of the function call. Using the parameters as arguments to the next recursive call allow the paths to be constructed through a systematic branch-and-bound type of search while maintaining current calculations of path weights and temporary starting location references.

```
begin route function

    /* passed: theLoc, thePath, theWeight */

    if theLoc = Vdest then begin
        set theWeight to 0
/* Vloc is previously assigned eval point */
        set VSubDist to
            Vloc path distance to Vdest
/* route only to another pt closer to dest */
        if VTotDist > VSubDist then begin
            for each Vloc in thePath do begin
                set theWeight to
                    Vloc current + theWeight
/* weight of path = number of claims */
            end
            if theWeight < Vweight then begin
                set Vweight to theWeight
/* lowest total weight of path to dest */
                set Vpath to thePath
/* list of locs along best path to dest */
            end
        end
        remove last object from thePath
        set Vpath to thePath
        return true
    end

/* AdjacentLocs returns all adjacent control
points from the current location */
```

```
        for each Vloc in AdjacentLocs(theLoc)
        do begin
            if Vloc = Vorigin then begin
                /* looped around to current loc */
                remove last object from thePath
                set Vpath to thePath
                return false
            end

            /* add loc to path & recurse again */
            insert Vloc into thePath
            call route(Vloc,thePath,theWeight)
        end

        remove last object from thePath
        set Vpath to thePath
        return true
    end
```

5 PERFORMANCE COMPARISON

The example model was first set to use AutoMod default parameters. Loads were introduced at a location on the loop every 20 seconds and sent to another location immediately preceding the starting point. Empty vehicles were simply told to loop around again. With this loading rate and a total of 50 vehicles, approximately half the vehicles were being loaded, which provided a good steady-state case as a baseline. AutoMod automatically routed all vehicles around the loop using the shortest overall path.

For comparison, the loaded vehicles were instructed to use the route function described in this paper. Empty vehicles continued to use the shortest path around the loop and therefore created congestion in those lanes. Loaded vehicles chose the least crowded alternate lane in each 4-lane set. Automatic delays were set up in the lanes based inversely on the number of claims on the lane (more claims lessened the delay time, thereby tending to balance the number of vehicles in each set of lanes).

Table 1 shows the dramatic improvement in load delivery times (since the load introduction rate was well below the capacity of the system, there was very little difference in the wait for pickup times). It is interesting to note that by improving the delivery time performance, overall vehicle utilization declines.

Although these results represent one specific layout and other operating assumptions, the same type of improvement would be expected in other larger, more complex path networks since avoiding congestion delays reduces average delivery time. As in any movement system, specific results would depend on the given layout and operating methods.

Table 1: Delivery Comparison

	Avg. Delivery Time	Std. Dev.	Vehicle Utilization**
Shortest Path (all vehicles)	8.12 min.	0.01	51%
Routing Loaded Cars	4.44 min.	0.04	37%

Delivery time (*) represents the time that vehicles were in the process of moving with a load – from the time of pickup until set down.

Vehicle utilization (**) is a simple comparison of the number of vehicle trips with loads compared to the total number of trips around the loop (empty vehicles are continually routed around the loop if no load is present for pickup).

6 CONCLUSION

Traffic management is a common issue in modeling path based movement systems. In many cases, only local decisions are required based on conditions immediately downstream of an intersection. In those cases, AutoMod provides two built-in mechanisms for modeling routing of vehicles.

A recursive method of analyzing possible paths may be useful where complex routing decisions must be made. Evaluation methods may be incorporated into a systematic search at each routing node of the path and use current traffic congestion or other preferential information to base decisions. Performance comparisons may then be made testing different route evaluation criteria.

AUTHOR BIOGRAPHY

MICHAEL NORMAN is a Senior Simulation Analyst for Brooks-PRI Automation - Planning and Logistics Solutions. He has over 14 years experience in simulation modeling using the AutoMod family of products. Michael holds a B.S. in Industrial Engineering from the University of Washington, is a Professional Industrial Engineer and a Senior Member of IIE. His email and web addresses are `<Mike.Norman@brooks-pri.com>` and `<www.brooks-pri.com>`.