Common Ground Vehicle Route Planning for Army Simulations

Dr. Paul W. Richmond, PE.
U.S. Army Engineer Research and Development Center
Geotechnical and Structures Laboratory
3909 Halls Ferry Road
Vicksburg, MS 39180
Paul.W.Richmond@erdc.usace.army.mil

Dr. Randy K. Scoggins
U.S. Army Engineer Research and Development Center
Geotechnical and Structures Laboratory
3909 Halls Ferry Road
Vicksburg, MS 39180
Randy.K.Scoggins@erdc.usace.army.mil

Burhman Q. Gates
U.S. Army Engineer Research and Development Center
Geotechnical and Structures Laboratory
3909 Halls Ferry Road
Vicksburg, MS 39180
Burhman.Q.Gates@erdc.usace.army.mil

Harold Yamauchi
Rolands & Associates Corporation
500 Sloat Avenue
Monterey, CA 93940
hmyamauc@nps.edu

Keywords:
Route Planning, Vehicles, OneSAF Test Bed (OTB), Simulations, Networks, Shape File, Battlespace Terrain Reasoning and Analysis (BTRA)

ABSTRACT: *An ERDC research program, "Common Maneuver Networks for Embedded Training, Mission Planning and Rehearsal" (CMN) seeks to allow Army simulations to use a ground vehicle maneuver network generated with Commercial Joint Mapping Toolkit (C/JMTK) and Battlespace Terrain Reasoning and Analysis (BTRA) products (BTRA is also an ERDC research program). Achievement of this goal would mean mission rehearsals could be carried out on the same vehicle maneuver network used for mission planning. Initially, the intent was to demonstrate the use of a BTRA maneuver network in the OneSAF Objective System (OOS) and, thus, we developed all the computer code in Java. A recent effort in support of a geospatial battle management language (geoBML) demonstration resulted in a need for an implementation of this code in the OneSAF Testbed Baseline (OTB), along with an understanding of current OTB route planning.*

The task involved merging an external routing network and corresponding cost factors in shape file format into the current OTB ground vehicle route planning scheme. Rather than implementing C++ code to reproduce the Java functionality in OTB, the GNU (Operating System) Compiled Native Interface (CNI) was employed to compile the Java code into native code and link with the most recent OTB C++ code. An analysis was first performed to identify which behavior; libraries, finite state machines, etc., in OTB would be affected. The principle difficulty to be overcome was related to the time required to find a route (on the order of minutes) and the relation between the Java virtual machine and the C++ processes which occurs when multiple routes are requested.

This paper describes the implementation of Java-based route planning software with a BTRA-generated maneuver network into OTB 2.5. Current route planning methods in OTB are also discussed, along with efforts associated with implementation into OOS and how the Java code used is also related to COMBATXXI development efforts.

1. Introduction

The Army's Future Combat System (FCS), a system of systems, and Future Force will depend heavily on a common operational picture (COP) of the battlespace. The FCS COP must be able to depict terrain and weather effects on operations for mission planning, rehearsal, and execution (battle command) as well as for training. Embedded training is a key component of FCS and will be used to explore and develop courses of action for decision support. To enable the Future Force to see first, understand first, act first, and finish decisively, FCS must have the capability to transition seamlessly between Battle Command (BC) systems and models and

simulations (M&S). Current Embedded Training and Battle Command systems do not share tactical maneuver data. The battlespace COP is, therefore, inconsistent between these systems, potentially leading to severe consequences from incorrect decisions about maneuver potential during training, planning, and execution of operations. The Battlespace Terrain Reasoning and Awareness (BTRA) ERDC research program provides tools to the Commercial Joint Mapping Tool Kit (C/JMTK) which will feed BC systems. There is potential for linking common software elements in both OOS and BTRA (e.g., Standard Mobility Application Programmer's Interface [1] and the logical environmental data model). The M&S scope of this research is limited to OneSAF Objective System and OneSAF Testbed Baseline (OTB). The BC scope is focused on C/JMTK via BTRA. The environment is limited to those features and attributes dealing with maneuver networks. Behaviors/battlespace functions are limited to those associated with ground vehicle maneuver. The overall objective of this research is to develop a common, consistent capability for assessing mobility and dynamic maneuver potential between BTRA and Army simulations.

This paper describes the implementation of route planning code into OTB. It describes our Java code and how it was linked to OTB using the GNU Compiled Native Interface (CNI). New and modified OTB code libraries are described. Comparisons are made between a standard OTB route and a BTRA based route, along with the additional environmental influences that can be considered.

2. Background

In force on force simulations, the principal goal of long range route finding (this paper is limited to discussion of long range planning as opposed to short range, which is concerned with dynamic obstacles such as other vehicles) is to be able to move to a given point avoiding obstacles. In more advanced route finding, the route can be optimized based on a criterion (on-road only, off-road, fastest, most concealed, etc). One way to do this is to develop a network graph which represents the maneuver environment and apply a cost to each edge or segment of the network. In the simplest case, the length of an edge is the cost, but costs can also be associated with the terrain and estimated vehicle speeds. The network data is stored in a forward star structure – the most efficient format for representing networks [2]. Mathematically, this is known as a shortest path problem, and two well known solution methods are Dijkstra's [3] and the A* [4] algorithms. There are other methods for route finding such as wavefront propagation in a visibility graph where obstacles are modeled as opaque. Wavefront propagation is resource and computation intensive. Bellman-Ford route finding allows negative edge costs but doesn't scale well.

The OneSAF Test Bed, with respect to vehicle performance modeling, is based on a relatively simple terrain model, and while extensions and other improvements [5-9] have increased the ability to affect vehicle speeds based on terrain conditions, the underlying network or "routemap" used by OTB for long range route planning appears unchanged from its earliest version as MODSAF. Figure 2.1 shows the obstacles and corridors displayed in an OTB plan view, along with a platoon route around an obstacle. These obstacles, corridors, and bypasses (not shown in the figure) are compiled by preprocessing the terrain database and creating a *.rnl file (where the * is the name of the terrain file specified). For the simple case of routing around an obstacle in an open area, the route planner works well. Routing through a complex area (e.g. mountain pass) can be problematic and depends on the fidelity of the terrain database.

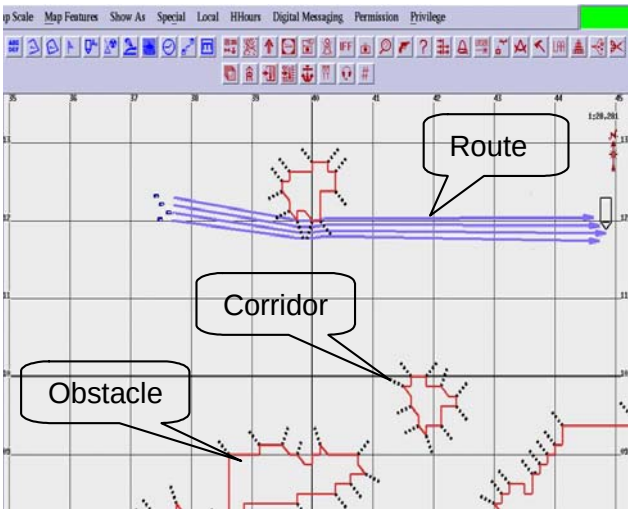

Figure 2.1 Obstacles and corridors from the OTB *.rnl file, terrain features not shown for clarity.

The Pathfinder project (formally titled: "Integration of Urban Characterization, Munitions Effects & Threat Assessment for Ground Vehicle Planning in Urban Environments") was an effort sponsored by the Battle Command Simulation Experimentation Directorate over several years. The target simulation for this project was the Combined Arms Analysis Tool for the XXIst Century (COMBAT^{XXI}). One of Pathfinder's objectives, "vehicle route planning in urban environments with consideration of movement rates", resulted in Java code for performing

vehicle route planning based on network graphs, which were used in this effort.

OneSAF Objective System (OOS) also makes use of a network graph for ground vehicle route planning. However, there are differences in network compilation. Part of the network is defined by the road network and compiled before simulation start. The other part of the network is implicit and is compiled at run-time from a grid overlaid over the cross-country terrain. However, the network representing cross-country terrain is expanded only as needed to find each route. OOS makes use of the A* algorithm to find least-cost paths through the network whether on-road, cross-country, or combined networks.

The Commercial Joint Mapping Toolkit[1] (C/JMTK) is the Department of Defense's future geospatial and visualization toolkit; it is a geospatial information system (GIS) based on ESRI products. The ERDC's Battlespace Terrain Reasoning and Analysis research project is producing tools which support ground vehicle mobility analysis and tactical maneuver planning. For actual unit route planning, it uses ESRI's proprietary network analyzer package, which utilizes the A* algorithm. A specific product of interest is a maneuver network which can be exported as a shapefile.

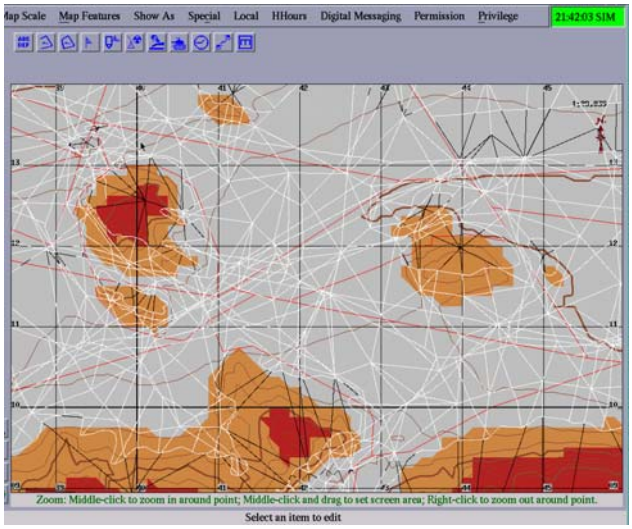

Figure 2.2 A BTRA network overlaying the OTB terrain.

The shapefile[2] format actually consists of a series of files:

- *.shp - holds the actual vertices of each geometric object.
- *.shx - holds index data pointing to the structures in the *.shp file.
- *.dbf - holds the geometric object attributes in dBase format.
- *.prj - an ascii text file that contains information about the map projection of the shapefile.
- *.sbn and *.sbx - hold spatial indices for use for read/write operations on shapefiles.

Figure 2.2 shows a BTRA network overlaying the OTB terrain; the network consists of nodes (end points) and arcs. Each arc has attributes based on mobility assessment of the corridor or road it represents. White lines are the trafficable arcs, black lines are not trafficable (obstacle areas). Although formal documentation of the BTRA network arc attributes is limited, currently it contains about 56 attributes [10] which can be used to develop a cost associated with finding a route, many of these are based on the vehicle-terrain interaction of the 12 vehicle classes represented in the standard mobility API [1].

3. Implementation

3.1 Overview

The OTB code is organized by libraries or directories with each library prefaced with "lib". Thus, for example, the libmove directory contains source code and documentation files relating to movement. The conceptual flow chart in Figure 3.1 shows our libmaneuvernet library as it relates to other parts of OTB. A command line argument is used to activate initialization and availability of our code to the simulation. The maneuver network shape files are expected to be in the same directory as the OTB terrain file and have the same name, with standard shape file extensions, as described above.

3.2 Libmaneuvernet

The major elements of the libmaneuvernet library are:

- **man_net_JVM** - The OTB C++ wrapper class used to execute PATHFINDER Java code. It was created using OTB's model template. It also transfers data to and from the Java Virtual Machine (VM) variable address space.
- **jman_net** - This is a Java object that is associated with and called by the **man_net_JVM** object. Its main purpose is to execute other PATHFINDER methods to compute a route.

[1] http://www.cjmtk.com/
[2] http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf

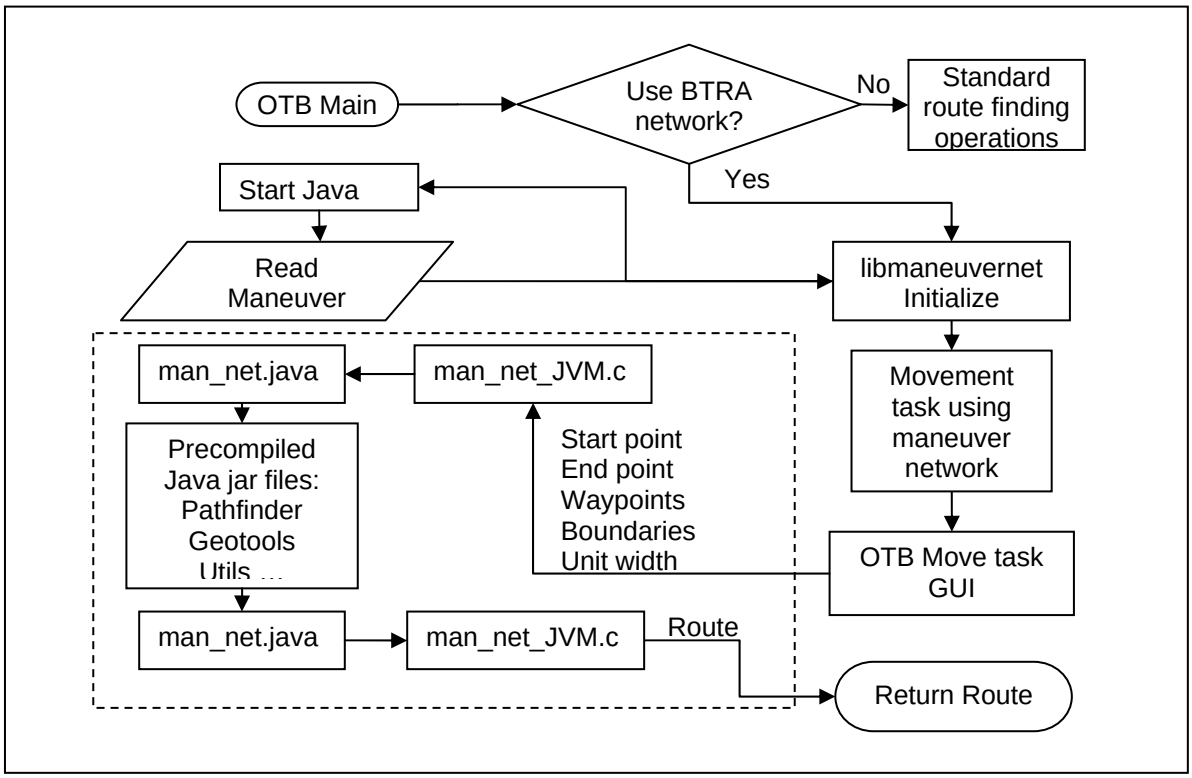

Figure 3.1 Flow chart of OTB/BTRA maneuver network implementation.

- **ext_jotb** - A sub directory of libmaneuvernet, which contains external Java archives (jar files). During the make process, these are copied to the OTB jar directory.

A **man_net_JVM**[3] C++ object and a corresponding **jman_net** object together act as the bridge between the C++ code and the Java source code. The **jman_net** object is an object inside the Java Virtual Machine (VM). However, **jman_net** also has a representation understood by C++ code. We accomplished this by using the GNU Java compiler gcj and the Compiled Native Interface (CNI) framework[4]. The gcj compiles the full PATHFINDER code base and produces jar files that are executed in the GNU Java VM. The GNU CNI, described in more detail below, allows gcj to produce extra files that allow a C++ class to define variables and methods that correspond to Java types. There is also a C++ interface to access the Java VM variables in a straight-forward way as well as the reverse. Although this approach requires use of the GNU compilers to work, we found the CNI to be robust and simpler to implement C++ code than Sun's Java Native Interface (JNI) which is more portable. The newly created library libmaneuvernet thus provides the bridge between C++ in OTB and the Java code in the PATHFINDER route planner. The libmaneuvernet performs two main functions: execute PATHFINDER code methods consisting of standard Java code, and transfer data between the C++ **man_net_JVM** object and the Java **jman_net** object. The meaning of data transfer here is to copy values between C++ memory and Java VM memory for use in the other language. The C++ names and a short description of the fields in **man_net_JVM** are given in Table 3.1. Those variables which start with "goal" are a reflection of the OTB routepoints structure, and not of all these values are currently used by the **jman_net**. The Java class **jman_net** includes corresponding fields within the definition of corresponding Java data types, as well as several methods employed to copy arrays to/from C++ pointers, to read terrain network data,and to compute the route by calling PATHFINDER methods. The **jman_net** methods are listed and described in Table 3.2. Two significant methods in the **man_net_JVM** C++ class are the initialization function **man_net_init** and the route planning function **man_net_getRoute.**

Initialization consists primarily of creating the Java VM

[3] Java and C++ class and method names are shown in bold type.
[4]The GNU Compiler for the Java Programming Language. http://gcc.gnu.org/java/

and instantiating the **man_net_JVM** object, which in turn creates an instance of the **jman_net** Java object. Java methods are then called to initialize and read the BTRA network from the shapefiles. The **man_net_getRoute** method takes two sets of coordinate points as arguments in addition to left and right boundary line segments. These are converted from OTB system coordinates to LAT-LONG coordinates and used to place a route request in the queue of the PATHFINDER routing service.

Table 3.1 Java class **man_net_JVM** fields for data transfer.

unitWidth	Unit formation or vehicle width
goal_num_pts	Number of points in the requested plan, at least two.
goal_mes_id	An array of multi-elevation structure ids associated with each goal_num_pts
goal_encl_id	An array of enclosure ids
goal_user_data	An array of user data
goal_point_id	An array of point ids, which points to the actual coordinates
goal_next_segment _width	An array of route widths associated each segment
goal_points	An array containing the route waypoints returned by PATHFINDER

Table 3.2 Java class **jman_net** methods.

jman_net	Class constructor
setShapefileBase PathName	Sets string to location of terrain network files
getNextArcWidth	Returns the width of an arc from network database
routemap	Copies the route request from C++, calls PATHFINDER, copies results from Java to C++ overwriting the passed arguments
checkEnabled	Builds indicators to display enabled/disabled BTRA network arcs in OTB
createNetwork	Builds PATHFINDER network files from the shapefiles
readNetwork	Reads in PATHFINDER network
getProjection	Retrieves the geospatial projection to use for coordinate conversions
readShapeLayer	Reads in the shape layer file

PATHFINDER calculations can be time consuming, particularly for longer routes. The original implementation of route planning in OTB required the thread calling the route planning code to block until the route was returned. This was not considered suitable for BTRA planning due to the lags introduced in updating the OTB processes imposed by lengthy PATHFINDER route calculations. Therefore, a client/server arrangement was implemented in **man_net_JVM** to allow the calling thread to continue while a PATHFINDER server computes the route. The libBTRAtraveling library[5] required modifications to allow non-blocking behavior while waiting for the client/server thread to finish. The required modifications amount to separating the route list creation processes into a portion performed before the request and a portion performed after the request had returned. The existing implementation blocked on each route request call. In the threaded applications, the calling thread, that is the spawned BTRA move task, is responsible for checking the libmaneuvernet **man_net_JVM** object periodically to determine if a new route is available for the vehicle id associated with the route request. The non-blocking **man_net_JVM** method **man_net_poll** is available to test the PATHFINDER server request to determine if a given route has been computed and is available. Note that the dashed outlined area in Figure 3.1 is the portion of the route finding task performed by the PATHFINDER server. Each request is polled at every scheduled tick for all vehicles to check the associated request ID before proceeding. The vehicle will continue waiting until all the segments of a request have been completed. At this point, the BTRAtravel finite state machine transitions to following the route (i.e. moving).

3.3 GNU compiled interface

The CNI allows Java source code to be compiled and placed in a library for linking with other object modules created by GNU C++. In addition, CNI provides methods, data structures, and a preprocessor application that facilitate writing C++ code to access Java variables in the Java VM as well as the reverse. Any Java classes that are to be shared are first expressed in a Java source file. This is then processed by the GNU gcjh utility to generate a C header file in which Java classes are expressed in terms of C++ classes. All primitive Java data types (int, double, etc.) have C++ type definitions (jint, jdouble, etc.). Arrays of Java primitive data types, as well as untyped memory, may also be created in the Java VM by the C++ method **JvMalloc**. This memory is allocated and deallocated by the Java garbage collector just as in normal Java code; of particular usefulness to us was the ability to execute other

[5] Nearly a direct copy of the original OTB libutraveling library.

Java class byte code within the **libgcj** virtual machine.

3.4 Other changes to OTB

Other changes to OTB included a modification to libtactmap to allow the BTRA network to be drawn as a single object rather than a set of OTB line objects. This greatly reduced overhead and allowed the BTRA maneuver network to be displayed quickly. Various changes were made which allow a correct OTB build under Fedora Core 5 Linux. Fortunately, the changes to libBTRAtraveling solved most problems since unit, mixed, and company level routing is done by spawning libBTRAtraveling tasks. A side effect is that units in a company proceed as soon as a route is available and not all at once. The delay between starting is probably going to be small as the units of the company are in the same general location. This could be addressed by modifying higher level unit routing or formation maintenance behaviors.

3.5 External jar files

Vehicle routing through the BTRA maneuver network is provided by a Java route planning service originally developed under the Pathfinder project. The service was written as an Application Programmer's Interface (API) to determine the least cost path over a network graph. It implements an A* algorithm to find the least cost path from a single source vertex to a single destination vertex on a directed graph. A minimum-priority queue data structure was implemented to support the algorithm.

The API was written as generically as possible. This means that no cost functions or special algorithms were written within the API to calculate edge costs. The idea was to allow the application developer to have the ability to write a library of cost functions with each cost function incorporating the appropriate algorithm. These cost functions can then be instantiated and the appropriate one passed to the API as the situation dictates. In order for the developer to accomplish this, the API provides a Java interface called **CostFunctionIfc** that declares three methods. The developer writes a cost function that "implements" all of the methods declared in **CostFunctionIfc.** For example, one of the methods declared in **CostFunctionIfc** is **calculateCost,** which returns the non-negative cost of traversing an edge. The developer must create a cost function class and write a **calculateCost** method that returns a cost based on the algorithm appropriate for the application. The cost must be non-negative since the A* algorithm assumes non-negative costs. Eight cost functions were created for this project that implemented the **CostFunctionIfc** interface: (1) arc distance, (2) arc traversal time, (3) arc distance where a penalty is enforced whenever an off-road edge is utilized, (4) arc distance where a penalty is enforced whenever a road arc is utilized, (5) edge traversal time where a penalty is enforced whenever an off-road arc is utilized, (6) arc traversal time where a penalty is enforced whenever a road arc is utilized, (7) cover/concealment, and (8) arc distance with a penalty for arc's which have a width less than the unit's width.

The route planning service API assumes the graph is represented as a collection of adjacency lists, and that arc flow is unidirectional. In addition, there are interfaces defined in the API associated with the arcs, nodes, and graph called, respectively, **ArcIfc**, **NodeIfc**, and **NetworkIfc**. The methods declared in these interfaces are needed to either support the bookkeeping performed by the A* algorithm (for example, to obtain the shortest path estimate from the source to the current vertex, a method called **getShortestPathEst** is declared in **NodeIfc**), or to simply access some component of the graph (for example, to return all of the vertices in the graph in a Java Collection, a method called **getNodes** is declared in **NetworkIfc**).

Since the BTRA maneuver network is exported in shapefile format, the *.shp file, in particular, is not structured as required by the route planning service API; i.e., the maneuver network is not represented as a collection of adjacency lists and arc flow is bi-directional. Since it could not be accessed directly by the route planning service, a class to convert the *.shp file into a format compatible for the API was developed. This class, called **NetworkGenerator**, makes use of a GeoTools library that can access *.shp files to generate an ascii file in the desired format. GeoTools[6] is an open source Java toolkit for software developers who need to manipulate geospatial data. It implements Open Geospatial Consortium (OGC)[7] specifications and provides a framework for developers to easily implement OGC-compliant server-side services, standalone applications or applets. GeoTools is under constant development (latest stable version is 2.2.2) and is released under the GNU Lesser General Public License (LGPL). In addition to providing access to the *.shp file, GeoTools is also used by all eight cost functions to access the .dbf file to obtain cost data.

[6] GeoTools home page: http://geotools.codehaus.org/
[7] Open Geospatial Consortium home page: http://www.opengeospatial.org/

Finally, an OpenMap™-based viewer was developed to help verify that the route planning service and the eight cost functions were implemented correctly. OpenMap™[8] is an open source JavaBeans-based geospatial toolkit by BBN Technologies. Like GeoTools, it is under constant development (latest stable version is 4.6.3) and is freely available, downloadable from the OpenMap™ website. The download has a viewer packaged as a sample application. Using the OpenMap™ API, we wrote our own layers and integrated them into the viewer in order to display the maneuver network, call the route planning service, and display the resulting route. Lastly, JDOM[9] was included as a dependency of GeoTools, but was not used directly.

4. Results

Figure 4.1 shows a platoon moving along a route based on the fastest route to the "Destination". Formation, vehicle spacing, and speed values are controlled by unchanged OTB methods. For platoon routing in OTB, one route is found and then, based on the formation, the 4 offset routes are created. If the width of the arc is not sufficient for the unit's formation, OTB will change the formation to column.

Figure 4.2 shows a similar movement goal, using the standard OTB route planner (long blue lines). The short pink lines are the result of the local map planner which seeks to avoid the obstacle not planned for in the initial route. Notice that the local planner has moved the vehicles off the original plan, and will eventually follow a path around the obstacle. Local map planning is still available to the vehicles following a BTRA route.

5. Future Work

Work in the near future will involve bounding the search area for the route with unit boundaries. Integrating PATHFINDER routing using BTRA networks in OOS is also planned for this year. Using a BTRA network for cross country maneuver planning sometimes results in zigzag routes, as movement is specified as lines directly between vertices. OTB entity movement behaviors need to allow movement across the entire edge width and also allow "cutting corners" when appropriate.

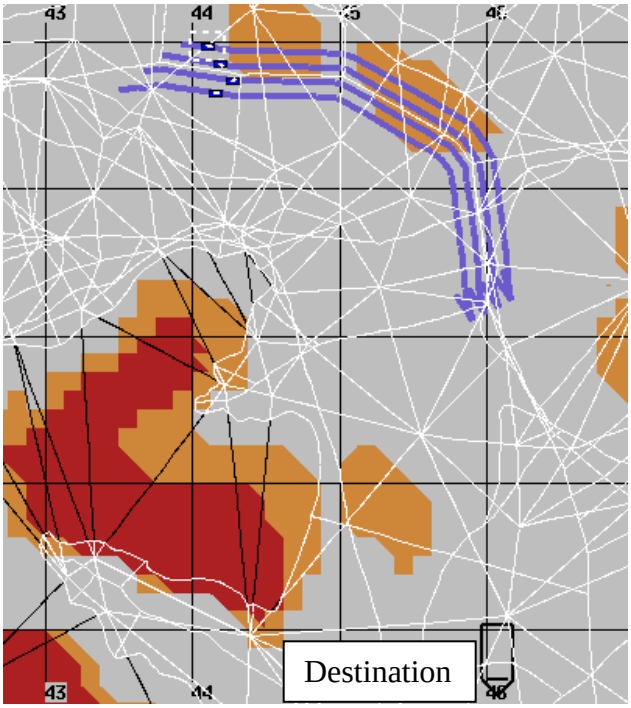

Figure 4.1 Fastest route for a platoon using the BTRA network in OTB.

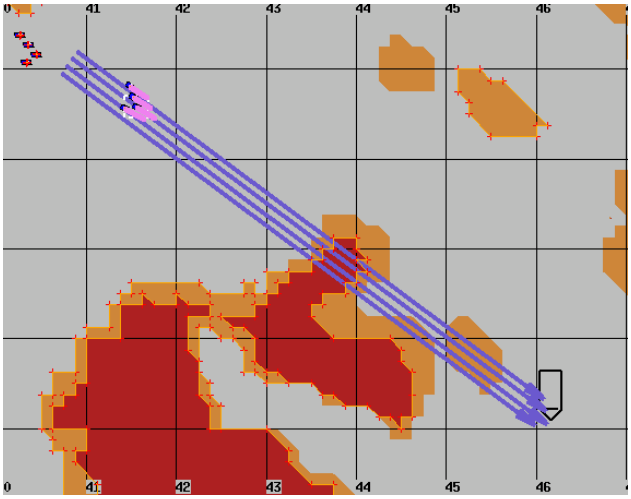

Figure 4.2 Platoon routing using standard OTB move task.

6. Summary

The purpose of this effort was to investigate the seamless transfer of maneuver potential between BC and M&S systems. In so doing, an order prepared in a system using BTRA components can be exercised on terrain with consistent ground vehicle movement potential. Implications are that as vehicles execute these tasks in the operations order, locally planned routes over consistent

[8] OpenMap home page: http://openmap.bbn.com/

[9] JDOM home page:: http://www.jdom.org/index.html

terrain will result in realistic movement, which takes into account recent terrain analysis and intelligence data.

6. References

[1] Baylot Jr., E.A., B.Q. Gates, J.G. Green, P.W. Richmond, N.C. Goerger, G.L. Mason, C.L. Cummins, and L.S. Bunch: "Standard for Ground Vehicle Mobility," U.S. Army Engineer Research and Development Center, Geotechnical and Structures Laboratory, Vicksburg, MS ERDC/GSL TR-05-6, 2005.

[2] Gallo, G., and S. Pallottino (1988) "Shortest Paths Algorithms" Annals of Operations Research, 13, 3-79.

[3] E.W. Dijkstra: "A Note on Two Problems in Connexion with Graphs" Numerische Mathematik, Vol1, pp. 269-271, 1959.

[4] R. Sedgewick and J.S. Vitter: "Shortest Paths in Euclidean Graphs" Algorithmica, Vol. 1, No. 1, pp. 31-48, March 1986.

[5] U.S. Army (1996) "CCTT Dynamic Behavior, Design Synthesis Report" PEO-STRI, Orlando, FL.

[6] U.S. Army (1996) "Compendium of CCTT Algorithms, Data, Data Structures and Generic System Mappings" PEO-STRI, Orlando, FL.

[7] Joint Precision Strike Demonstration (JPSD) Program Office (2002) "System Detailed Description for the Joint Virtual Battlespace (JVB)" Fort Belvoir, VA.

[8] Mason, G.L., R.B. Ahlvin, and J.G. Green (2001) "Short-term Operational Forecasts of Trafficability" ERDC/GSL TR-01-22, U.S. Army Engineer Research and Development Center, Vicksburg, MS.

[9] Condon, P. (2002) "Routing Design Notes V0.1 for the OneSAF ERC Program" Science Applications International Corporation.

[10] Blais, C.L., N.C. Goerger, P. Richmond, B. Gates, and M. Pace (2005) "Data Mapping and Ontology Design for Common Maneuver Networks" Paper 05S-SIW-140, Spring 2005 Simulation Interoperability Workshop, San Diego, CA.

7. Acknowledgment and Disclaimer

This research was sponsored by the U. S. Army Engineer Research and Development Center, Geotechnical and Structures Laboratory. Permission was granted by the Director, Geotechnical and Structures Laboratory to publish this information.

Author Biographies

PAUL W. RICHMOND Ph.D., P.E. is a mechanical engineer at the U.S. Army Corps of Engineers, Engineer Research and Development Center (ERDC) where he develops ground vehicle mobility models for use in simulations, simulators and performance analysis models, specifically related to terrain interaction and off-road performance He obtained his Bachelor of Science from Clarkson University, a M.S. from Dartmouth College, and his Doctorate from the University of Alaska, Fairbanks.

RANDY K. SCOGGINS Ph.D. is a physicist at the U.S. Army Corps of Engineers, Engineer Research and Development Center (ERDC) where he develops models for use in simulations, simulators and analysis, specifically related to visualization and performance optimization. He obtained his Bachelor of Science from Mississippi College, a M. S. from Georgia Tech, and his Doctorate in Computational Engineering from Mississippi State University.

BURHMAN GATES began his career at the U.S. Army ERDC in 1990. He works in the areas of testing, modeling, and transfer of vehicle terrain interaction models. He is a member of SAME, IEEE, and the National Society of Professional Engineers and has served as president of the Mississippi Engineering Society Vicksburg Chapter. Mr. Gates received his B.S. degree in Electrical Engineering from Louisiana State University.

HAROLD YAMAUCHI is a programmer with Rolands & Associates Corporation supporting the U.S. Army TRADOC Analysis Center – Monterey. He received his A.B. degree in Statistics from the University of California, Berkeley, and his M.S. degree in Operations Research from Oregon State University.