COMPUTER SIMULATION OF INDIVIDUAL AND GROUPED MILITARY OBJECTS REDEPLOYMENT

ZBIGNIEW TARAPATA

Institute of Mathematics and Operations Research
Faculty of Cybernetics, Military University of Technology
Kaliskiego Street 2, 00-908 Warsaw
Tel. (022)-685-94-13, Fax. (022)-685-75-39
e-mail : ztarap@isi.wat.waw.pl

A method for redeployment simulation of military objects is considered. An optimization problem of simultaneous arriving of all redeployed objects to some points is defined and the method of its solving is shown. Definitions of objects needed to making simulation in object-oriented environment are made known. Methods of redeployment simulation of individual and grouped objects based on notation of MODSIM II object-oriented simulation language are proposed.

Introduction

Redeployment simulation of military objects is a basic problem in combat simulators. Objects movement is very important from the point of view of simulating complex system. It may have an effect on accuracy, adequateness, effectiveness and other characteristics of these systems. This problem applies not only in military fields but also, for example in: transport, telecommunication traffic, computer networks, etc.

The objects movement problem requires algorithms or methods suitable to move various objects. The algorithms may base on: Dijkstra algorithm [3], algorithms for determination of the N-th shortest path ([4], [5], [6], [7]), algorithm for determination of the shortest K-path ([11], [13]) or others ([1], [2], [8]). Result of objects movement algorithms will be the set of paths on which we will simulate objects movement. In such a case objects movement plan will be known in advance. But in real combat actions a random modification of path may be needed in some cases:

- sudden change of decision by commands;
- change of the network status (destruction of a node or arc belonging to the path for some objects);

Other problem is an appropriate description of the terrain area for which the movement is carried out. In many cases, the model of it is a network. Depending on: types of moved objects, preferences of movement methods (individual, grouped) and military actions level we must build the network based on, e.g. numerical terrain model ([12], [14]).

Taking into consideration the description presented above the purpose of this paper is to solve the problem of movement simulation of objects which may have various groups with constraints on parallel movement of objects' head.

The paper is organized as follows.
In section **1** a model of network in which the movement is carried out is defined. Section **2** contains definition of the problem of simultaneous arriving of all redeployed objects to some fixed points. In section **3** and **4** methods for individual and grouped objects movement simulation, respectively, are presented.

1. Model of network

The idea of a network creation (called „trafficable routes network") for military objects movement was presented in [12]. Using this, we can describe the network **S** as follows:

$$\mathbf{S} = \langle G, D \rangle \tag{1}$$

where:

G – Berge graph, without loops, describing the structure of **S**,

$G = \langle W_G, U_G \rangle$;

$D = [d_{w,w'}]_{W \times W}$ - matrix of distances between the graph nodes, (2)

W= $|W_G|$; W_G - set of the graph nodes, U_G - set of the graph arcs.

Let the maximum speeds of K moving objects be v^1, v^2, ... , v^k, ... , v^K, respectively. Because these speeds may be various for each of the objects, so the vector $t_{w,w'}$ of minimum crossing times of $u_{w,w'}$ arc assumes the following form :

$$t_{w,w'} = \langle t^1_{w,w'}, t^2_{w,w'}, \dots, t^k_{w,w'}, \dots, t^K_{w,w'} \rangle, \quad w, w' \in W_G \tag{3}$$

where:

$$t^k_{w,w'} = \begin{cases} \dfrac{d_{w,w'}}{v^k \cdot \zeta(u_{w,w'})}, & u_{w,w'} \in U_G \\ 0, & w = w' \\ \infty, & u_{w,w'} \notin U_G \end{cases}, \quad k = \overline{1,K}, \tag{4}$$

The function $\zeta(\cdot)$ values are in the interval (0,1], and it describes coefficient of speed decrease in topographical or weather conditions, etc., for a part of the road which represents this arc. In that case, network **S** from (1) may be described as follows :

$$\mathbf{S}' = \langle G, t_u \rangle \tag{5}$$

where :

$$t_u = [t_{w,w'}]_{W \times W} \tag{6}$$

and $t_{w,w'}$ is described by (3).

We will utilize network **S** (1) for finding the shortest paths in the sense of terrain distances, whereas network **S**' (5) will be used for finding the shortest paths in the sense of time.

2. Definition of the objects alignment problem

In practice, during redeployment of many objects we use the so-called alignments lines which are utilized to align of the moving objects heads, that is to guarantee a relative parallelness of objects movement. We solve the problem defined below.
We denote

$$I_k = \left(i^0(k), i^1(k), \dots, i^r(k), \dots, i^{R_k}(k) \right) \tag{7}$$

$$T_k(I_k) = T_k = \left(\tau^0(k), \tau^1(k), \dots, \tau^r(k), \dots, \tau^{R_k}(k) \right) \tag{8}$$

where :

I_k - set of nodes describing the path for the k-th object;

$i^r(k)$ - the r-th node on the path for the k-th object;

T_k - set of time instances of achieving the nodes belonging to the path for the k-th object;

$\tau^r(k)$ - time instance of achieving node $\mathbf{i}^r(k)$ by the head of the k-th object, $\underset{k=\overline{1,K}}{\forall}\ \underset{r=\overline{0,R_k-1}}{\forall} \tau^{r+1}(k) \geq \tau^r(k)$;

R_k+1 - number of nodes belonging to the path of k-th object.

Let

$$I_k^{'} = \left\{ i_1^{'}(k), i_2^{'}(k), ..., i_p^{'}(k), ..., i_{\mathbf{P}_k}^{'}(k) \right\} \tag{9}$$

where :

$i_p^{'}(k)$ - the p-th element of $I_k^{'}$ satisfying $\underset{p=\overline{1,\mathbf{P}_k}}{\forall} i_p^{'}(k) \in I_k$,

denote the set of nodes, at which we must align the head of the k-th object in relation to the heads of other objects.

Let, by analogy

$$T_k^{'} = \left\{ \tau_1^{'}(k), \tau_2^{'}(k), ..., \tau_p^{'}(k), ..., \tau_{\mathbf{P}_k}^{'}(k) \right\} \tag{10}$$

denote a set of demanded time instances of arriving to particular alignment nodes by the k-th object head.

We make the following assumption :

$$\mathbf{P}_1 = \mathbf{P}_2 = ... = \mathbf{P}_K$$

that is, for all objects exists the same number of alignment points (nodes).

At this point we can define the problem as follows :

$$\min \sum_{k=1}^{K} \sum_{p=1}^{\mathbf{P}_k} \left(\tau_p^{'}(k) - \tau_p(k) \right) \tag{11}$$

with the constraints :

$$v_{r,r+1}^{'k} \leq v_k, \qquad r = \overline{0, R_k - 1},\ k = \overline{1,K} \tag{12}$$

$$v_{r,r+1}^{'k} > 0, \qquad r = \overline{0, R_k - 1},\ k = \overline{1,K} \tag{13}$$

where :

$\tau_p^{'}(k)$ - an element of (10);

$$\tau_p(k) = \tau^0(k) + \sum_{\{r \in \mathbf{R_k} : r \leq r^{'}\}} t_{r,r+1}^{'k}; \tag{14}$$

$$r^{'} = r \in \mathbf{R_k} : i^r(k) = i_p^{'}(k); \qquad \text{(compare with (8))} \tag{15}$$

$$t_{r,r+1}^{'k} = \frac{v^k}{v_{r,r+1}^{'k}} \cdot t_{i^{r^{'}}, i^{r^{'}+1}}^{k}; \tag{16}$$

$$\mathbf{R_k} = \{0,1,...,r,...,R_k\}. \tag{17}$$

We see that speed $v_{r,r+1}^{'k}$ of movement on the arcs belonging to the path of the k-th object will be a decision variable for each of the objects. Solution to this problem is carried out in two stages. Firstly, we must determine shortest paths for K objects (e.g. using the algorithm from [13]) and fix I_k and T_k sets. Secondly, we solve a hiperbolic programming problem described by (11) ÷ (13) using one of the well-known methods (e.g. ellipsoidal algorithm).

Constraint (13) may be replaced by

$$v_{r,r+1}^{'k} \geq v_{\min} \quad , v_{\min} > 0 \tag{18}$$

if we want to have a lower constraint on the speed of the k-th object.

The more practical problem may be one when we have a set of alignment nodes for each of the objects and want to achieve appropriate alignment nodes at the same time by heads of K objects. We solve this problem by modification of the objective function (11). Let

$$\bar{\tau}_p = \frac{\sum_{k=1}^{K} \tau_p(k)}{K} \quad , p = \overline{1, \mathbf{P}_k} \tag{19}$$

denote the average of time instances of reaching the p-th alignment nodes by K objects, and

$$\hat{\tau}_p = \frac{\sum_{k=1}^{K} \left(\bar{\tau}_p - \tau_p(k)\right)^2}{K} \quad , \ p = \overline{1, \mathbf{P}_k} \tag{20}$$

denote the variance of time instances of reaching the p-th alignment nodes by all objects.
Then, we modify the objective function (11) :

$$\mathbf{min} \sum_{p=1}^{P_k} \hat{\tau}_p \tag{21}$$

with the constraints (12) and (13) or (12) and (18).
This problem may be useful, for example, in the case of movement armoured detachments, when we want to move tanks in an equal line. At that time, we find the shortest paths for K objects and determine the alignment nodes for each of the paths and find an arc speed of each of the objects by solving the problem described by (21) with constraints (12) and (13).

3. Method for movement simulation of individual objects

Movement simulation of military objects will be carried out in the environment of simulation object oriented language MODSIM II. Therefore, we will consequently use the notation of this language.

Each of the military objects will be considered as a separate MODSIM object. For MODSIM objects utilized by the object type „military object” we define the following :

- **VehicleObj** = OBJECT
  - *nr* : INTEGER; (* object number *)
  - *nr_nad* : INTEGER; (* number of superior unit *)
  - *v_max* : INETEGR; (* maximal speed *)
  - *rodz* : BOOLEAN; (* object type: TRUE -centipeded, FALSE- vehicular)
    -
    - other fields (see attributes vector of military unit in [12])
    -

  ASK METHOD *SetFields*(IN nr, nr_nad, v_max : INTEGER; . . .);
  ASK METHOD *ObjInit*();
  END OBJECT;

- **Wsp** = RECORD
  x, *y*, *z* : REAL;
  END RECORD;

- **NodeObj**=OBJECT(**ImageObj**, **QueueObj**)
  Translation : PointType;
  Nr : INTEGER;

```
                •
                •       other methods defining the node
                •
    END OBJECT

•   LinkObj=OBJECT(ImageObj);
        Source, Destination : NodeObj;
                •
                •       other fields and methods defining the network link (arc)
                •
    END OBJECT;

•   NetworkObj = OBJECT
        NrOfNodes : INTEGER;                (* number of nodes *)

        ASK METHOD GiveLink(IN node1, node2 : NodeObj): LinkObj;
        ASK METHOD GiveNode(IN nr : INTEGER) : NodeObj;
                •
                •       other methods defining the network
                •
    END OBJECT;

•   DynVehicleObj = OBJECT(VehicleObj, DynImageObj)
        Course, Speed : REAL;         (* inherited from MovingObj *)
        MovingTo : BOOLEAN;           (* inherited from  MovingObj *)

        RotationSpeed : REAL;         (* inherited from RotatingObj*)
        RotatingTo : BOOLEAN;         (* inherited from  RotatingObj*)

        ScaleSpeed : REAL;            (* inherited from ScalingObj *)
        ScalingTo : BOOLEAN;          (* inherited from ScalingObj *)

        Motion : BOOLEAN;             (* inherited from DynamicObj*)

        Translation : PointType;      (* inherited from GraphicVObj*)
                •
                •       other fields inherited from superior objects
                •
        Path : ARRAY INTEGER OF  INTEGER;  (* the field added by this object *)
        CurrNode : NodeObj;                                           .

        ASK METHOD SetCourse(IN course : REAL);      (*inherited from MovingObj*)
        ASK METHOD SetSpeed(IN speed : REAL);                         .
        TELL METHOD MoveTo(IN XDest, YDest : REAL);                   .
        TELL METHOD FollowPath(IN path : PoinArrayType);              .

        ASK METHOD SetRotationSpeed(IN rotSpeed : REAL);
                                                     (* inherited from RotatingObj*)
        TELL METHOD RotateTo(IN theta : REAL);                                 .
        ASK METHOD SetScaleSpeed(IN scaleSpeed : REAL);
                                                     (*inher. from ScalingObj*)
        TELL METHOD ScaleTo(In xScale, yScale : REAL);                         .

        ASK METHOD StartMotion;
                                                     (*inher. from DynamicObj*)
        ASK METHOD StopMotion;                                                 .
        ASK METHOD DynamicUpdate(IN currTime, elapsedTime : REAL);             .

        ASK METHOD SetCurrNode(IN node : NodeObj);
                                                     (* methods added by this object*)
        ASK METHOD SetPath(IN path : ARRAY INTEGER OF INTEGER);
        ASK METHOD FindPath(IN nr_wpocz, nr_wkon: INTEGER;
                            IN net : NetworkObj): ARRAY INTEGER OF INTEGER;
        TELL METHOD MoveVehicle(IN NodeS, NodeD : NodeObj);
                •
                •      other methods inherited from superior objects
                •
        ASK METHOD ObjInit();
    END OBJECT;
```

VehicleObj object contains attributes of military object, as information indispensable considering terrain traffic possibility by this object, etc.
Record **Wsp** contains information about coordinates. **NodeObj** and **LinkObj** objects contain definitions of the network node and arc, respectively. **NetworkObj** object defines the network containing, among other things, information about network nodes (coordinates and size of the node) and links.
DynVehicleObj object describes a military object containing, additionally, possibility of moving and imaging, and inherit both from **VehicleObj** and **DynImageObj**.
DynImageObj object ([9], [10]) is the standard object of SIMGRAPHICS II and describes dynamic graphical object :

```
    DynImageObj = OBJECT(ImageObj, MovingObj, RotatingObj, ScalingObj);
         •
         •      fields and methods (see [13], pp. 192-194)
         •
    END OBJECT;
```

This object may be drawed, moved, scaled and rotated with respect to simulation time. In this connection **DynVehicleObj** object have the same properties because inherited from **DynImageObj**.
The most important properties of this object we present below.

Properties of **DynVehicleObj** object

- inherited from **MovingObj** :
  fields
    * *Course* - actual course (direction) of object in radians in the world coordinate system;

  * *Speed* - object speed in the world coordinate units per time unit;
  * *MovingTo* - TRUE if object is actually moving;

  methods
  * ASK METHOD *SetCourse(...)* - sets direction which the object will travel;
  * ASK METHOD *SetSpeed(...)* - sets the speed of object;
  * TELL METHOD *MoveTo(...)* - moves the object to a specified point. The method stops when the object arrives at the destination.
  * TELL METHOD *FollowPath(...)* - moves the object along a path defined by the array of points. This method stops when the object arrives at the last point in the array. To stop it from continuing use *Interrupt*.

- inherited from **RotatingObj** :

  fields
  * *RotationSpeed* - actual speed of rotation in radians per seconds;
  * *RotatingTo* - TRUE if object is actually rotating;

  methods
  * ASK METHOD *SetRotationSpeed(...)* - sets the speed of rotation in radians per second. Negative values cause clockwise rotation;
  * TELL METHOD *RotateTo(...)* - rotates the object by angle in radians. Does not stop the execution of the program but is carried out synchronically with other simulation methods;

- inherited from **ScalingObj** :

  fields
  * *ScaleSpeed* – actual speed of object scaling;
  * *ScalingTo* - TRUE if object actually scaling;

  methods
  * ASK METHOD *SetScaleSpeed(...)* – sets the amount that is added to an object scaling factor every time unit. For example, with the scale of 1.0, object will become twice as big after 1 time unit, 3 times as big after 2 time units, etc.;
  * TELL METHOD *ScaleTo(...)* – synchronic scaling of the object to the point defined as the method parameter with speed *ScaleSpeed*;

- inherited from **DynamicObj** (which inherited from **MovingObj**, **RotatingObj**, **ScalingObj)** :

  fields
  * *Motion* - TRUE if object is currently moving;

  methods
  * ASK METHOD *StartMotion* - starts an object movement. After the method is invoked the *DynamicUpdate* method (described below) will be called automatically from the runtime library;
  * ASK METHOD *StopMotion* - stops an object from moving. *DynamicUpdate* method will no longer be invoked from the runtime library;
  * ASK METHOD *DynamicUpdate(IN currTime, elapsedTime : REAL)* - called periodically by the timing routine to update animation. The current simulation time is passed with the simulation time elapsed since the last *DynamicUpdate* call.

Animation (moving) of **DynImageObj** object type can be done in two ways. First of the ways is to set of object fields *Course* and *Speed* and invoke the *StartMotion* method of this object. It causes object movement with fixed attributes. Second of the ways is to use TELL or WAIT FOR instructions for the TELL method (e.g. *MoveTo*, *ScaleTo*, *RotateTo*), which causes time elapsing and synchronous invoking TELL methods which are stopped after reaching a destination point.

The fields and methods addded by **DynVehicleObj** are the following :

- field *CurrNode*;
- field *Path* ;
- ASK method *SetCurrNode(...)* ;
- ASK method *FindPath(...)* ;
- TELL method *MoveVehicle(...)*.

CurrNode field contains information about the network node lastly achieved by the object. *Path* field contains array of nodes numbers belonging to the path for the current object.
SetCurrNode(...) method is invoked when the object achieves the next node on its path.
FindPath(...) method that sets path for an object. Result is an array of node numbers belonging to the path from the starting node to the ending one for the current object .
TELL method *MoveVehicle(*IN *NodeS, NodeD : NodeObj)* causes synchronous movement of the object from *NodeS* to *NodeD*. This is the most important method from the point of view of movement simulation. Possible code of it is presented in Example 1.

<u>*Example 1*</u>

```
TELL METHOD MoveVehicle (IN NodeS, NodeD : NodeObj);
  VAR
        link            : LinkObj;
        NetWindow       : NetworkObj;
        i               : INTEGER;
        xd,xs,yd,ys     : REAL;
        exit            : BOOLEAN;

  BEGIN

   ASK SELF TO DisplayAt(ASK NodeS Translation.x, ASK NodeS Translation.y);
   WHILE (i < > HIGH(Path)+1) AND (NOT exit)
     INC(i);
     IF  i < HIGH(Path)
       link := ASK NetWindow TO GiveLink(
               ASK NetWindow TO GiveNode(ASK SELF  Path[i]),
               ASK NetWindow TO GiveNode(ASK SELF  Path[i+1]));
       IF link <> NILOBJ

       { object moving }

         NodeD := ASK link Destination;
         NodeS := ASK link Source;
        xs:=ASK NodeS Translation.x;
         ys:=ASK NodeS Translation.y;
         xd:=ASK NodeD Translation.x;
         yd:=ASK NodeD Translation.y;
         ASK SELF TO SetRotationSpeed(RotationSpeed);
         WAIT FOR SELF TO RotateTo(ATAN2(ys-yd,xs-xd)+pi);
         ON INTERRUPT
           IF SELF<>NILOBJ
             DISPOSE(SELF);
           END IF;
           exit:=TRUE;
         END WAIT;
```

```
21         ASK SELF TO SetSpeed(Speed);
22         WAIT FOR SELF TO MoveTo(xd, yd);
23         ON INTERRUPT
24           IF SELF<>NILOBJ
25             DISPOSE(SELF);
26           END IF;
27           exit:=TRUE;
28         END WAIT;
29       END IF;
30     END IF;
31   END WHILE;

32  END METHOD;
```

*

In line 14 the rotation with a fixed angle is done. In line 21 the object speed on the arc from *NodeS* to *NodeD* is set. This speed may be known by solving of the problem described in section **3**. Invoking of the method to start a synchronous object movement to the specified point (node) is presented in line 22. Independently of it objects may be moved by means of *StartMotion* method (see description earlier presented).

The full invoking of object movement may look like this :

Example 2

```
             .
             .
             .
 VAR
       vehicle          : DynVehicleObj;
       NetWindow        : NetworkObj;
       path             : ARRAY INTEGER OF INTEGER;

 BEGIN
       NEW(vehicle);
             .
             .
             .
       path:=ASK vehicle TO FindPath(NrOfStartingNode,NrOfEndingNode,NetWindow);
       ASK vehicle TO SetPath(path);
       nodeS:= ASK NetWindow TO GiveNode(NrOfStartingNode) ;
       nodeD:= ASK NetWindow TO GiveNode(NrOfEndingNode) ;
       TELL vehicle TO MoveVehicle(nodeS, nodeD);
       StartSimulation();
             .
             .
             .
       StopSimulation();
```

*

4. Method for movement simulation of grouped objects

A method of movement simulation for grouped objects is strictly related to the movement of individual objects. As example of grouped object may be column (convoy) of individual objects. In this case, movement of these objects may look like below:

Example 3

```
             .
             .
             .
 VAR
       VehicleColumn            : ARRAY INTEGER, INTEGER OF VehicleObj;
       ColumnsNumbers,
          HowManyInColumn        : INTEGER;
          delayTime              : REAL;

  BEGIN
          NEW(VehicleColumn,1..ColumnsNumbers,1..HowManyInColumn);
          FOR i:=1 TO ColumnsNumbers
            FOR j:=1 TO HowManyInColumn
              NEW(VehicleColumn[i,j]);
               path:=ASK vehicle TO FindPath(NrOfStartingNode+i,
                                               NrOfEndingNode+j,NetWindow);
              ASK VehicleColumn[i,j] TO SetPath(path);
              nodeS:= ASK NetWindow TO GiveNode(NrOfStartingNode+i) ;
              nodeD:= ASK NetWindow TO GiveNode(NrOfEndingNode+j) ;
              TELL VehicleColumn[i,j] TO MoveVehicle(nodeS,nodeD) IN delayTime;
            END FOR;
           END FOR;

          StartSimulation();
                    .
                    .
                    .
           StopSimulation();
```

*

Using the instruction „TELL VehicleColumn[i,j] TO MoveVehicle(nodeS,nodeD) IN delayTime" causes that particular objects of column will follow previous object (that is second behind the first, third behind the second, etc.) with a delay equal to *delayTime*. Value of this delay may be changed and then we can use the *StartMotion()* and *DynamicUpdate()* methods to dynamic changing of path for each object.

Summary

A movement of grouped objects may be carried out in the other way. We must find for objects group the shortest K-path ([13]) or for each element of the group the shortest path ([1], [2], [3], [4], [5], [6], [7]) to the destination point (node) and next we invoke *MoveTo()* or *FollowPath()* methods for each element of each objects group. These methods are interrupted in the case of : decision changing by commands or destroying of the network elements belonging to the path for some object. At that time we must determine the shortest paths from the last achieved points by objects for particular elements of the group and repeat invoking of the *MoveTo()* or *FollowPath()* methods, or *StartMotion()* and *DynamicUpdate()* ones.
Solution of the problem defined in the section **2** and idea of various object movement simulation have been utilized in MODSIM application obtained after realization of the research work described in [14] to animation and movement simulation of military vehicles. Obtained solutions may be useful in transport scheduling, visualization and animation and the like applications.

References

[1]. Cai X., Kloks T., Wong C.K. : *Time-varying shortest path problems with constraints,* Networks **29** (1997), 141-149.

[2]. Denardo E.V., Fox B.L. : *Shortest-route methods: 1. Reaching, pruning and buckets,* Operations Research **27** (1979), 215-248.

[3]. Dijkstra E. : *A note on two problems in connection with graphs*, Numerische Mathematik **1** (1959), 269-271.

[4]. Dreyfus S. E. : *An appraisal of some shortest path algorithms*, Operations Research **17** (1969), 395-412.

[5]. Golden B.L., Skiscim C.C.*: Solving k-shortest and constrained shortest path problems efficiently*, Network Optimization and Applications **20** (1989), Texas A&M University, College Station.

[6]. Hoffman W., Pavley R. : *A method for the solution of the Nth best path problem*, Assoc. Comput. Mach. **6** (1959), 506-514.

[7]. Ibaraki T., Katon N., Mine H. : *An O(Kn^2) algorithm for K shortest simple paths in an undirected graph with nonnegative arc length*, Trans. Inst. Electron. and Comm. Eng. Jap. **12** (1978), 1199-1206.

[8]. Kaszubowski Z., Mizera R., Piasecki S. : *Problemy przegrupowania wojsk*, Wojskowa Akademia Techniczna, Warszawa 1970.

[9]. *MODSIM II. The Language for Object-Oriented Programming. Reference Manual*, CACI Products Company, 1994.

[10]. *Simgraphics II. User's Manual for MODSIM II*, CACI Products Company, 1995.

[11]. Tarapata Z. : *Algorytmy komputerowego wspomagania planowania przemieszczania równoległego kolumn*, Rozprawa doktorska, Wojskowa Akademia Techniczna, Warszawa 1998.

[12]. Tarapata Z. : *Modelling of terrain for necessities of military objects movement simulation*, Bulletin of Military University of Technology **6/7** (1998), (in the press).

[13]. Tarapata Z. : *Algorithm for simultaneous finding a few independent shortest paths*, Conference Proceedings of 9th European Simulation Symposium, pp.89-93, Passau 1997.

[14]. Tarapata Z. : *Symulacja przemieszczania środków walki – pojedynczych, grupowych i oddziałów*, w opracowaniu wewnętrznym WAT z pracy badawczej dofinansowywanej przez KBN z grantu nr 0S001 015 07 pt. : „Komputerowa symulacja działań bojowych", Wojskowa Akademia Techniczna, Warszawa 1996.

Komputerowa symulacja przemieszczania pojedynczych i grupowych środków walki

Przedstawiono metodę symulacji przemieszczania środków walki. Zdefiniowano problem równoczesnego dotarcia przemieszczanych obiektów do pewnych ustalonych punktów oraz zaproponowano metodę jego rozwiązania. Przedstawiono definicję obiektów w środowisku zorientowanym obiektowo, które wykorzystywane są przez program symulacyjny. Zaproponowano metody symulacji przemieszczania pojedynczych i grupowych środków walki używając notacji zorientowanego-obiektowo języka symulacyjnego MODSIM II.