Zbigniew Tarapata

Models and Algorithms for Knowledge-Based Decision Support and Simulation in Defence and Transport Applications

```
WHILE Q ≠ null DO
  u := Extract_Min(Q);
  /u is such node that
  d[u] = min{d[v]: v∈Q}/
  Q := Q \ {u};
  IF u=t THEN
    RETURN;
  END IF;
  FOR EACH arc (u,v)∈A
    starting from u DO
  IF d[v]>d[u]+c(u,v) OR
     (d[v]=d[u]+c(u,v) AND
       p[v]<p[u]*q(u,v,d[v]))
  THEN
     d[v]:= d[u] + c(u,v);
     p[v]:= p[u] * q(u,v,d[v]);
     predecessor[v]:= u;
  END IF;
```

Zbigniew Tarapata

Models and Algorithms
for Knowledge-Based Decision Support and Simulation
in Defence and Transport Applications

Warsaw (Poland) 2011

Reviewer: *Prof. Andrzej Ameljańczyk*

Command and control processes in the defence domain are very complex with information of intensive activities, involving many variables with strong interrelationships and uncertainty. Two of the factors, which are especially essential in military decision-making are human battlefield stress and limited time. Therefore, it is very important to give, for decision-makers, computer tools, which support their decisions, simulate effects of these decisions and try to partially eliminate negative impact of their stress on decision being made, shorten the decision-making time and improve the degree of training.

The main goal of this book is to present new and analyse existing models and algorithms for decision support and simulation, especially in defence and transport applications, in a knowledge-based environment. The works have mainly focused on computational complexity and accuracy of presented algorithms as well as their usefulness in existing and new computer-based decision support and simulation systems. The integration of presented models and algorithms with computer tools causes that decision making in complex situations may be easier, faster and more effective than without a computer. Many of the presented models and algorithms are interdisciplinary, but in a majority of the cases we focus on defence applications.

We present models and algorithms for: terrain-based paths planning (decomposition, multiresolution, multicriteria and disjoint), movement synchronization scheduling, automatization of selected decision processes (identification of decision situations with distance vectors and multicriteria weighted graphs similarity approaches, decision automata to a march) as well as movement simulation of individual and cooperating group objects.

Selected applications of presented models and algorithms in real systems such as: *Zlocien*, *Guru*, *MSCombat*, *CAVaRS* have been discussed.

English proofreading: *Wojciech Gilewski*

Cover design: *Barbara Fedyna*

Edition I, Warsaw 2011

ISBN 978-83-62954-15-5

Publisher: Wojskowa Akademia Techniczna
00-908 Warszawa, ul. S. Kaliskiego 2

Printed by: P.P.H. Remigraf
ul. Ratuszowa 11, 03-450 Warszawa

Acronyms

Acronym	Description
2CMSS	*2-Criteria Movement Synchronization Scheduling*
ADAMS	*Allied Deployment and Movement System*
BST	*Binary Search Tree*
C2	*Command and Control*
C3	*Command, Control and Communications*
C3I	*Command, Control, Communications and Intelligence*
C4ISR	*Command, Control, Communications, Computers, Intelligence, Surveillance and Reconnaissance*
CAVaRS	*Course of Action Verification and Recommendation Simulation System*
CAX	*Computer Assisted Exercises*
CBS	*Corps Battle Simulation*
CCTT	*Close Combat Tactical Trainer*
CGF	*Computer Generated Forces*
CoA	*Course of Action*
DAG	*Directed Acyclic Graph*
DEM	*Data Exchange Mechanism*
DIS	*Distributed Interactive Simulation*
DP	*Disjoint Paths*
DSP	*Decomposition Shortest Paths*
DSS	*Decision Support System*
FIFO	*First In First Out*
FPTAS	*Fully Polynomial Time Approximation Scheme*
GAMS	*General Algebraic Modelling System*
Guru	*System of Automatic Tools for Decision Support – Guru*
HA	*Hierarchical Algorithm*
HLA	*High Level Architecture*
HSP	*Hybrid Shortest Path*
JC3IEDM	*Joint Consultation, Command and Control Information Exchange Data Model*
JTLS	*Joint Theatre Level Simulation*
KB	*Knowledge Base*
*LRTA**	*Learning RTA**

MIP	*Multilateral Interoperability Program*
ModSAF	*Modular Semi-Automated Forces*
MOSP	*Multiobjective (Multicriteria) Shortest Paths*
MPM	*Movement Planning Manager*
MS	*Movement Synchronization*
MSA	*Movement Synchronization Algorithms*
MSCombat	*Modelling & Simulation System of Combat*
MSM	*Movement Synchronization Manager*
MSS	*Movement Synchronization Scheduling*
MSSD	*Movement Synchronization Scheduling with Distance*
MSST	*Movement Synchronization Scheduling with Time*
MWGSP	*Multicriteria Weighted Graphs Similarity Problem*
NDRP-Max	*Node Disjoint Restricted Paths* minimize maximal cost
NDRP-Sum	*Node Disjoint Restricted Paths* minimize total cost
NDSP	*Node Disjoint Shortest Paths*
PDM	*Pape-D'Esopo-Moore algorithm*
PDSP	*Parallel Decomposition Shortest Paths*
PIMTAS	*Predictive Intelligence Military Tactical Analysis System*
PRDS	*Pattern Recognition of Decision Situations*
QoS	*Quality of Service*
RPLUM	*Route Planning Uncertainty Manager*
RSPP	*Restricted Shortest Paths Problem*
*RTA**	*Real-Time A**
RTEF	*Real-Time Edge Follows*
SAF, SAFOR	*Semi-Automated Forces*
SATDS Guru	*System of Automatic Tools for Decision Support – Guru*
SBOTTS Zlocien	*Simulation-Based Operational Training Support System – Zlocien*
SGDP	*Subgraphs Generating Disjoint Paths*
SPT	*Shortest Paths Tree*
SSA	*Simulation System Architecture*
TDN	*Time-Dependent Network*
TIN	*Triangulated Irregular Network*
TMA	*Tactical Movement Analyzer*
VPF	*Vector Product Format*
Zlocien	*Simulation-Based Operational Training Support System – Zlocien*

CONTENTS

1. Introduction

1.1. Research Domain

1.1.1. A Short Description

Decision making is an inseparable element of human life. Many of human decisions concern complex problems solving. These problems have properties, which distinguish them from simple problems (Pohl *et al.*, 2003): they can involve many related issues or variables; some of the variables may be only partially defined and some may yet to be discovered; complex problem situations are pervaded with dynamic information changes; solution objectives may change; they typically have more than one solution. The solution of complex problems can be categorized as intensive information activity, which its success depends largely on the availability of information resources and, in particular, the experience and reasoning skills of the decision-makers. This clearly presents an opportunity for the useful employment of computer-based *Decision Support Systems* (*DSS*) in which the capabilities of the human decision-maker are complemented with knowledge bases, expert agents, and self-activating conflict identification and monitoring capabilities. Therefore, we can write the following definition of the *DSS* (Holsapple & Whinston, 1996):

"The *Decision Support System* (*DSS*) is a computer-based information system that supports business or organizational decision-making activities".

In general, in the decision making process the following stages are considered (Najgebauer, 1999a): recognition of a decision situation, determination of possible decision variants, decision choice, estimation of effects of decisions being realized, modification or changing the decision.

Each of these stages can be supported by a computer. A computer support causes that decision making may be easier, faster and more effective than without a computer. Several models and methods from such domains as operations research (e.g. simulation, optimization, games theory, etc.), pattern recognition, transportation (e.g. paths planning), analysis of algorithms are used. Each of these methods can be supported by a computer as well.

One of the most complicated and complex decision processes concerns military applications. Much has been written in literature about the complexities of planning and execution of these processes (Dockery & Woodcock, 1993; Ground *et al.*, 2002; Moffat, 2003; Najgebauer, 1999a; Pohl *et al.*, 2003; Przemieniecki, 1994;

Sawyer, 1995). Military command and control processes are information intensive activities, involving many variables (tasks of friendly forces, expected actions of opposite forces, environmental conditions – terrain, weather, time of the day and season of the year, current state of own (friend) and opposite forces in the sense of personnel, weapon systems and military materiel, etc.) with strong interrelationships and uncertainty. Two of the factors which are especially essential in military decision-making are human battlefield stress and a limited time. Therefore, it is very important to give, for military decision-makers, computer tools, which support their decisions and try to partially eliminate the negative impact of their stress on the decision being made and shorten the decision-making time. Moreover, the information sources are typically widely distributed and subject to continuous change. In such a case, in order to improve situational awareness, data fusion and integration is done (Antkiewicz *et al.*, 2010b; 2010d; Chmielewski, 2008a; Chmielewski & Kasprzyk, 2008b; Koszela & Chmielewski, 2008; Najgebauer *et al.*, 2008d; Smart *et al.*, 2005).

A typical military decision planning process is similar to a general decision making process described earlier and it contains the following steps:

1. the assessment of both own and opposite forces, terrain as well as other factors which may have an influence on a task realization,
2. the identification of a decision situation,
3. the determination of decision variants (*Course of Actions*, *CoA*),
4. the variants (*CoA*) evaluation (verification),
5. the recommendation of the best variant (*CoA*) chosen among these that meet the proposed criteria.

One of the methods which can be used in the military decision planning process is computer simulation (Najgebauer, 1999a). Simulators are used in the following steps of this process: (4) the variant verification (via simulation) and (5) the variant recommendation. Moreover, simulation can also be used for:

- optimization of command chains of military units,
- evaluation of the military operational rules and improving the command and control procedures,
- research of the military equipment's parameters, which modify results of military actions,
- quality verification of battlefield process models (shooting, target searching, movement, etc.).

In other words: simulation results can be used to make or change decisions. On the other hand, simulation is one of the basic methods in military trainings (Najgebauer, 1999a). This is the second main role of the simulation in the military area.

One of the most important decision problems in the military area (but not only in this area) is movement planning. Object movement is an essential element of combat actions and it is related to manoeuvre planning of military detachments on the battlefield during battle as well as during preparation for battle. This process is very important from the point of view of simulating a complex system. It may have an effect on accuracy, adequacy, effectiveness and other characteristics of these systems. Redeployment planning and simulation of military objects is a basic problem, especially in combat simulators. Moreover, movement (paths, routing, motion) planning is also an essential element in other applications: civilian transportation, mobile robots, car navigation, virtual agents in computer games, etc. These properties make this problem as interdisciplinary and multi-domains. This problem should be solved using specialized algorithms to avoid its internal complexity (Tarapata, 2003b). Therefore, movement planning and simulation models and algorithms are one of the main problems considered in this book.
In the military domain, decision support and simulation systems can support systems of class *C4ISR* (*Command, Control, Communications, Computers, Intelligence, Surveillance and Reconnaissance*) and their types[1] (Pohl *et al.*, 2003; Ground *et al.*, 2002). In order to make better decisions, these systems should be a knowledge-based (*KB*). Models and algorithms for these two fields: decision support and simulation in *KB* environment are the most interesting from this book's point of view.

1.1.2. Knowledge-Based Decision Support

As it has been written in the previous chapter, the *Decision Support System* (*DSS*) should be a knowledge-based system. In this context *knowledge* can be described as (Pohl *et al.*, 2003):

"(...) experience derived from observation and interpretation of past events or phenomena, and the application of methods to past situations. Knowledge bases capture this experience in the form of rules, case studies, standard practices, and typical descriptions of objects and object systems that can serve as prototypes. Problem solvers typically manipulate these prototypes, in several different ways. Therefore, we use our knowledge of past similar situations as a baseline for defining the current problem system and developing a solution strategy (...)".

An example of a knowledge-based decision support system schema for military applications (borrowed from *Guru* system (Guru, 2005)) is presented in Fig. 1.1. We can observe two elements, which contain a knowledge base (*KB*): operational-tactical *KB* and terrain *KB*. The first one is used to collect knowledge being used to express the character of the digital battlefield during automation of

[1] *C2=Command and Control*; *C3I=Command, Control, Communications and Intelligence*, etc.

military decision-making: military rules, decision situation patterns and recognition rules, course of action (*CoA*) patterns, etc. The second one (terrain *KB*) collects pre-processed information from the terrain database. For example, in chapter 2.3 we presented a network model of the terrain (with rule-based functions described on the network's nodes and arcs) in the *Zlocien* simulation system, which is based on pre-processed information from the terrain database, and in chapters 5.2-5.3 we use the operational-tactical *KB* to identify decision situations and automatization of the march process.

Other examples of knowledge-based decision support systems in the military area can be found in (Ground *et al.*, 2002; Pohl *et al.*, 2003).

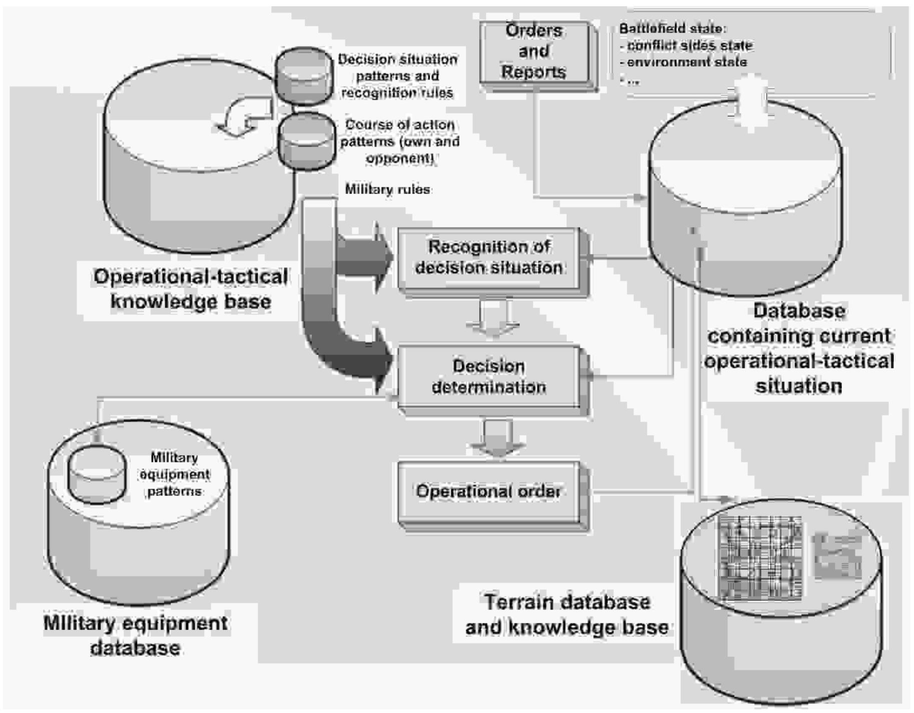

Decision situation patterns and recognition rules
Course of action patterns (own and opponent)
Military rules
Orders and Reports
Battlefield state:
- conflict sides state
- environment state
- ...
Operational-tactical knowledge base
Recognition of decision situation
Decision determination
Operational order
Database containing current operational-tactical situation
Military equipment patterns
Military equipment database
Terrain database and knowledge base

Fig. 1.1. An example of a knowledge-based decision support system schema for military applications (Guru, 2005)

For paths planning as one of the main elements in terrain(knowledge)-based decision support and simulation systems we can indicate many examples of knowledge-based applications: for mobile robots (Guo *et al.*, 2010; Heero, 2006; Hodal & Dvorak, 2008; Hu *et al.*, 2004; ; Nagarajan & Raja, 2010; Stentz, 1994; Weng *et al.*, 2009; Zafar *et al.*, 2006) and for the military (Campbell *et al.*, 1995; Gilmore & Semeco, 1985; Lee & Fishwick, 1995; Lee, 1996; Logan & Sloman, 1997b; Rajput & Karr, 1994).

1.1.3. Knowledge-Based Simulation

The knowledge-based simulation was conceived at the RAND Corporation in the late 1970's and early 1980's applying artificial intelligence to simulation

(Rothenberg *et al.*, 1989). One of the applications has been considered deals with knowledge-based validation of simulation results in the military domain (Kornell, 1987; Madni *et al.*, 1987). Other applications (civilian) of knowledge-based simulation have been described in (Cheung *et al.*, 2007; Duran & Costaguta, 2009; Oren, 2001; Robinson *et al.*, 2005; Zeigler *et al.*, 1991; 1996). Simulation is used, because a *knowledge-based simulation is understood as a compilation of simulation and artificial intelligence techniques*, hence it is agent-based (Oren, 2001). The agent simulation allows simulation of natural or engineered entities with cognitive abilities. Therefore, agent simulation is very appropriate for the simulation of intelligent entities. Agent-based simulation is the use of agent technology to generate the behaviour of models. There are many applications of agent-based simulation in the military area. In the paper (Reece, 2003) the author has developed a movement behaviour model for soldier agents who populate a virtual battlefield environment. Paper (Cil & Mala, 2010) proposes a two-layer hybrid agent architecture to match the needs of future multi-dimensional warfare. This architecture has an integrated simulation tool to simulate planning results from the cognitive layer via reactive agents. In the paper (Zafar *et al.*, 2006) the authors show the possibility of using hybrid architecture that implements mine detection, obstacle avoidance and route planning with a group of autonomous agents with coordination capabilities. Groups of inter cooperating multi agents working towards a common goal have the potential to perform a task faster and with an increased level of efficiency then the same number of agents acting in an independent manner. The paper (Montana *et al.*, 2000) discusses the proof-of-concept of an automated system for scheduling all the transportation for the United States military down to a low level of detail. Their approach is to use a multi-agent society with each agent performing a particular role for a particular organization. They show that the usage of a common multiagent infrastructure allows easy communication between agents, both within the transportation society and with external agents generating transportation requirements. In the paper (Gelenbe *et al.*, 2004) authors describe how a complex and simulation environment can be used to fuse information about the behaviour of groups of objects of interest. The fused information includes the objects' individual pursuits and aims, the physical and geographic setting within which they act, and their collective social behaviour. The group control algorithms combine reinforcement learning, social potential fields and imitation. The paper (Sahin *et al.*, 2008) deals with bio-inspired computation techniques, such as genetic algorithms, for real-time self-deployment of mobile agents to carry out tasks similar to military applications. In the paper (Wang, 2006) authors build stochastic mathematical models, in particular G-network models of behaviour. They have demonstrated their approach in the context of urban military planning and analyzed the obtained results. The results are validated against those obtained from a simulator. The results suggest that the proposed approach has tackled one of the classical problems in modelling multi-agent systems and is able to predict the systems' performance at low computational cost.

For many years in military applications a simulated battlefield is used for training military personnel. There are at least three ways to provide the simulated opponent:

- two groups of trainees in simulators may oppose each other (often used);
- human instructors who are trained to behave in a way that mimics the desired enemy doctrine (seldom used);
- computer system that generates and controls multiple simulation entities using software and possibly a human operator.

The last approach is known as a *Semi-Automated Force* (*SAF* or *SAFOR*) or a *Computer Generated Force* (*CGF*). The *CGF* is used in military *Distributed Interactive Simulation* (*DIS*) systems to control large numbers of autonomous battlefield entities using computer equipment and software rather than humans in simulators.

The advantages of the *CGF* are well-known (Petty, 1995):

- they lower the cost of a *DIS* system by reducing the number of standard simulators that must be purchased and maintained;
- *CGF* can be programmed, in theory, to behave according to the tactical doctrine of any desired opposing force, and so eliminate the need to train and retrain human operators to behave like the current enemy;
- *CGF* can be easier to control by a single person than an opposing force made up of many human operators and it may give the training instructor greater control over the training experience.

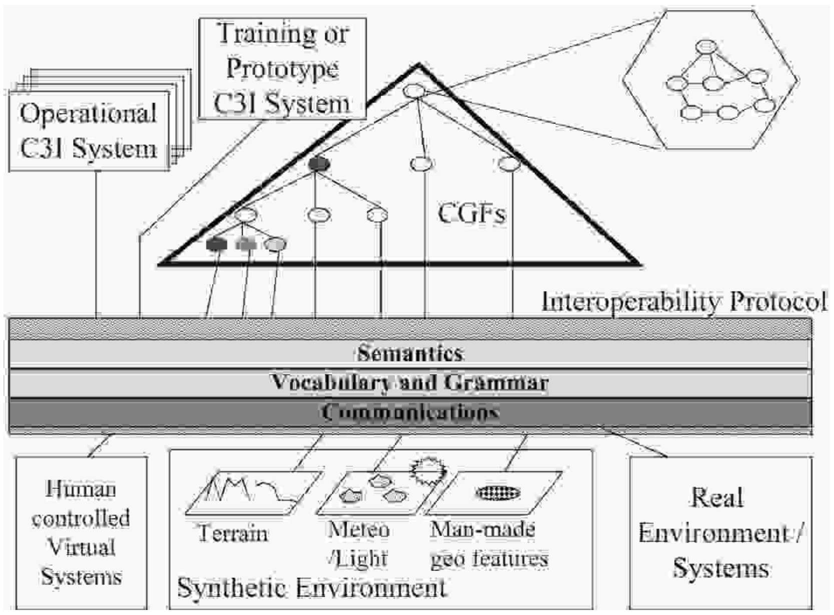


Fig. 1.2. A potential simulation system architecture for military applications (Dompke, 2001)

A potential simulation system architecture (*SSA*) and the location of the *CGF* system inside the *SSA* is presented in Fig. 1.2. Note that *CGF* can and should cooperate with *C3I* systems.

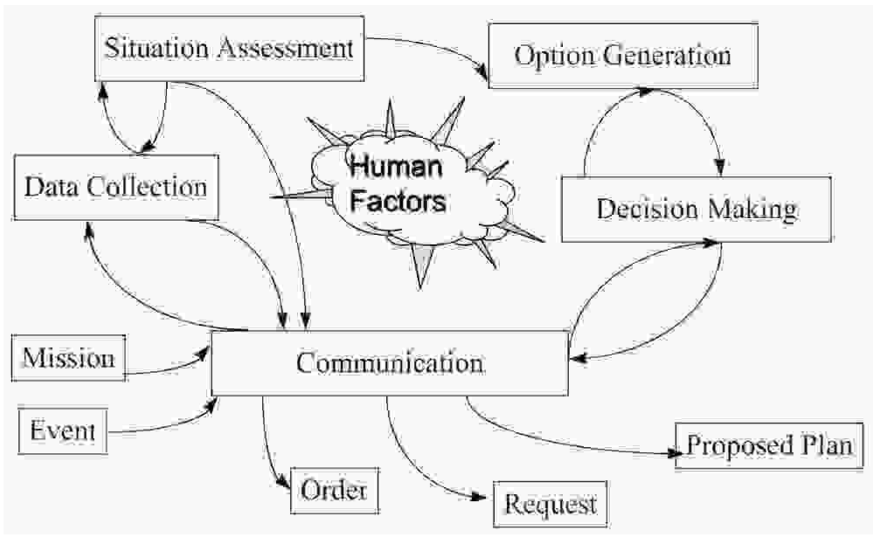


Fig. 1.3. Modules of the *CGF* rational/cognitive model (Dompke, 2001)

The role of each of the modules in *CGFs* (see Fig. 1.3) can be described as follows (Dompke, 2001):

(1) the *Data Collection* module is responsible for gathering the detailed data elements as instructed by the situation assessment module. Basic functions of this module are as follows:

(1.1) get data request,

(1.2) find data,

(1.3) prepare data,

(1.4) provide data reference;

(2) the *Situation Assessment* module defines the detailed data requirements that need to be collected, interprets the mission received by the *CGF*, updates the current assessment of the situation and defines and monitors critical and meaningful events. Basic functions of this module are as follows:

(2.1) produce data requests,

(2.2) interpret and fuse data,

(2.3) monitor critical events,

(2.4) maintain an updated situation;

(3) the *Option Generation* module develops courses of action (*CoA*) based on the triggering event, mission statement and current situation assessment. Basic functions of this module are as follows:

(3.1) generate possible courses of action;

(4) the *Decision-Making* module evaluates the various courses of action and ranks them according to a set of pre-determined and derived criteria. It will also support the negotiation process between *CGFs* or human decision makers that may be required to develop a solution for the larger context in which the *CGFs* decision are included. Basic functions of this module are as follows:

(4.1) rank options,

(4.2) goals decision making approach,

(4.3) negotiate;

(5) the *Communication* module supports the exchange of data between the *CGF* and all other elements of the simulation system. It transforms data into the appropriate format for local and external interpretation. Basic functions of this module are as follows:

(5.1) interface,

(5.2) report.

Selected technologies (important from our point of view) which are used by functions of *CGFs* are as follows (function number – technology (criticality: (L)ow, (M)edium, (H)igh)): (1.2) – knowledge discovery (L), knowledge based system (L), pattern recognition (L); (2.1) – knowledge discovery (H); (2.2) – pattern recognition (H); (3.1) – search algorithms (H), knowledge based system (H), models and methods of operations research (L); (4.1) – models and methods of operations research (H); (4.2) – planning algorithms (H), search algorithms (H).
From the description presented above results, that the *CGF* systems are strongly knowledge-based and they use models and methods of operations research.

As an inseparable part of the *CGF*, modules for route planning based on the real-terrain models are used (Ceranowicz, 1994; Dompke, 2001; Henninger *et al.*, 2000; OneSAF, 2008; Tuft *et al.*, 2006). For example in *ModSAF* (*Modular Semi-Automated Forces*), in module "SAFsim", which simulates the entities, units, and environmental processes the route planning component is located (Longtin & Megherbi, 1995).
Moreover, automated route planning will be a key element of almost any automated terrain analysis system that is a component of military *C4ISR* systems.

1.2. Research Objectives and Theses

The main goal of this book is to present new and analyse existing models and algorithms for decision support and simulation, especially in defence and transport applications, in a knowledge-based environment. The works have mainly focused on computational complexity and accuracy of presented algorithms as well as their usefulness in existing and new computer-based decision support and simulation systems. Many of the presented models and algorithms are interdisciplinary but in the majority of cases we focus on defence and transport applications.

It should be emphasized that the goal of this book ***are not*** problems concerning knowledge representation, knowledge acquisition, knowledge discovery, building expert systems, etc., but providing the models and algorithms to support the decisions and simulate their effects in *DSSs* and simulation systems, which are knowledge-based and the described algorithms can use this knowledge.

The main research theses presented in the book are as follows:

T1. knowledge-based decision support and simulation are effective methods to support decisions and simulate their effects in a dynamically changing environment and can be used in defence and transport applications; knowledge may concern an environment as well as decision processes being analysed;

T2. automatization of decision processes allow us to research these processes (e.g. using simulation) as well as decrease the cost and the time of complex process analysis;

T3. the use of specialized algorithms (which are new or adapted from existing algorithms) for solving decision problems can decrease computational complexity and/or increase accuracy of traditional algorithms; these algorithms can and should be a part of the knowledge-based *DSSs* and/or simulation systems.

These theses are verified in chapters of this book, which are organized as presented below.

1.3. Contents of the Book

Presented in chapter 2 is the review of methods of environment modelling for knowledge-based decision making and simulation. A few methods of digital map representation are described: the visibility diagram, Voronoi diagram, straight-line dual of the Voronoi diagram, edge-dual graph, line-thinned skeleton, regular grid of squares, grid of homogeneous squares coded in a quadtree system (as a representation of multiresolution terrain). An example is described of the terrain knowledge-based model being used in the real simulation *Zlocien* system. Moreover, four main approaches concerning terrain representation that are used in a battlefield simulation for paths planning have been described: free space analysis, vertex graph analysis, potential fields and grid-based algorithms.

Movement (paths, routing, motion) planning is an essential element in many applications: transportation, mobile robots, car navigation, virtual agents in computer games, military route planning, etc. Therefore, chapter 3 contains

a detailed discussion on three main models and algorithms for terrain-based paths planning: (1) decomposition and multiresolution approach to path planning, (2) multiobjective (multicriteria) paths planning and (3) disjoint paths planning.
In the first case, a decomposition method (*DSP* – decomposition shortest paths) is presented and its properties, which decrease computational time of path searching in multiresolution and large graphs. The goal of the method is not only computation time reduction but, most of all, using it for multiresolution path planning. A theoretical and experimental analysis of the method is discussed, especially from the computational complexity and accuracy point of view. The parallelization method of the *DSP* algorithm is also analysed. An example of using this method in a multiresolution battlefield simulation is described.
In the second case, selected multicriteria (multiobjective) approaches for the shortest path problems are presented. Classification of the multiobjective shortest path problems (*MOSP*) is given. Different models of *MOSP* problems are discussed in details. Methods of solving formulated optimization problems are presented. Analysis of complexity of presented methods and ways of adaptation of classical algorithms for solving multiobjective shortest path problems are described. The GAMS model for one of the *MOSP* problems is defined. Comparison of effectiveness of solving selected *MOSP* problems defined as: mathematical programming problems (using CPLEX 7.0 solver) and multi-weighted graph problems (using modified Dijkstra's algorithm) is given. Experimental results of using the presented methods for multicriteria path selection in a terrain-based grid network are given.
In the third case, specific disjoint paths planning models and algorithms are considered. We classify disjoint-paths planning problems and formulate two types of problems of node-disjoint paths visiting specified nodes: *NDRP-Sum* and *NDRP-Max*. The first one (*NDRP-Sum*) minimizes the total cost of all (K>1) disjoint paths visiting specified nodes in the restricted area and the second one (*NDRP-Max*) minimizes the maximal cost of any of the K disjoint paths. Exemplified GAMS models for both problems are defined. For solving the *NDRP-Sum* and *NDRP-Max* problems we propose the subgraphs generating-based algorithm (*SGDP*). Some experimental results with a discussion of the complexity and accuracy of the algorithm are shown in detail. Moreover, we show how to use modifications of the Busacker-Gowen and Edmonds-Karp minimal-cost flow algorithms to solve problems of the node-disjoint paths case.
Presented applications and examples of methods being described concern military applications, but these methods are interdisciplinary.

Chapter 4 deals with models and algorithms for the nonlinear optimization problem of multi-objects movement scheduling to synchronize their movement (*MS* problem) as well as properties of the presented algorithms. For synchronous movement, two categories of criteria are defined: the time of movement and "distance" of K>1 moved objects from the movement pattern. Similarities and differences between defined problems and the classical tasks scheduling problem in parallel processors are shown. Two algorithms for synchronous movement scheduling are proposed and their properties are considered. One of the algorithms is based on the dynamic programming approach and the second one uses approximation techniques. Moreover, we formulated the multicriteria movement synchronization scheduling problem (2*CMSS* problem). The model consists of two parts: (1) node-disjoint path planning visiting specified nodes for K objects with a given vector of intermediate nodes for each one (*NDSP* problem); (2) movement synchronization in intermediate nodes (*MS* problem). We defined the problem as a discrete-continuous, nonlinear, two-criterion mathematical programming problem. We proposed to use a two-stage algorithm to solve the 2*CMSS* problem (as a lexicographic solution): at first we have to find the vector of node-disjoint shortest paths for K objects visiting intermediate nodes to set optimal paths under the assumption that we use maximal possible velocities on each arc belonging to a path for each object (the solution of the *NDSP* problem using algorithms described in chapter 3), and next we try to decrease values of velocities to optimize the second criterion (synchronization, solution of the *MS* problem using algorithms described in this chapter). Theoretical and experimental analysis of the complexity and accuracy of the algorithms as well as their practical usefulness are discussed.

In chapter 5 the idea and model of command and control processes applied to the decision automata for attack, defence and marching on the battalion level as well as methods for movement simulation of individual and group objects are considered. The decision automata being presented replace battalion commanders in some simulator for military trainings and it executes two main processes: decision planning process and direct combat (or march) control. One of the elements of the decision planning process is an identification of the decision situation. Therefore, the model of the decision situation and two algorithms for decision situation identification are presented. The first one is based on a distance vector approach and the second one – on a multicriteria weighted graph similarity approach (*MWGSP* problem). Some numerical examples have been described. In the decision automata to march, the march planning process (containing: march organization determination and detailed march schedule determination) and the direct march control (containing: march simulation, identifying fault situations during a march simulation and automata reactions, velocity calculations and fuel consumption calculation) as well as techniques regarding automata implementation have been presented. Moreover, methods for movement simulation of individual and group objects based on the MODSIM simulation language have been discussed. These discussions are supplemented by the presentation of the method for cooperating objects movement simulation and management in real simulation system like *CGFs*.

Chapter 6 contains selected applications of described models and methods in real systems. We showed applications of methods described in chapters 3, 4 and 5 for movement planning and simulation in *Simulation-Based Operational Training Support System - Zlocien* and *Modelling & Simulation of Combat - MSCombat*. Next, we presented using these methods in knowledge-based pattern recognition tools to support military mission planning and simulation (in systems *Guru* and *CAVaRS*). Additionally, applications of the presented models and methods in security (especially in early warning systems) and crisis management systems are discussed.

Finally, remarks and conclusions concerning the described models, algorithms and related problems are presented.

1.4. Authorship and Bibliography Remarks

The author of this book is the author of the majority of presented models and algorithms. Authorship concerns: all models and algorithms presented in chapters 2.3, 3, 4, 5 and 6.1 (excluding: (1) computer implementation of the *SGDP* and *DSP* algorithms in chapter 3 – these implementations have been done by two supervised students; (2) the model of the decision situation in chapter 5.2.1 and (3) the method described in chapter 5.2.2). In the remaining cases the author of this book is the co-author. The majority of these models and algorithms are used in real systems: *Zlocien*, *Guru*, *MSCombat*, *CAVaRS*. These applications are described in separate chapters, especially in chapter 6.

Detailed state of the art and bibliography concerning problems presented in the book are discussed in suitable chapters. However, the fundamental sources of information for the newest research results were the following scientific journals and conference proceedings: *Computers & Operation Research*, *Networks*, *Journal of the ACM*, *Lecture Notes in Computer Science*, *Lecture Notes in Artificial Intelligence*, *Conference on Computer Generated Forces and Behavioural Representation*, *IEEE Computational Intelligence for Security and Defence Applications Conference*, *Military Communication and Information Systems Conference* and technical reports from selected research projects in which the author of this book has participated as a member or as the project manager: (Antkiewicz *et al.*, 2000; 2009d; Guru, 2005; Tarapata, 1999c; 2008f; 2010h; Zlocien, 2002).

In this book we consequently use separate notations in each chapter. However, in some cases we use the same notations as were previously used – in such a case we refer to these ones.

2. Environmental Modelling for Knowledge-Based Decision Making and Simulation

2.1. An Overview

The terrain database-based model is being used as an integrated part of the military *DSS* and *CGF* systems as well as in civilian applications. Terrain data can be as simple as an array of elevations (which provides only a limited means to estimate mobility) or as complex as an elevation array combined with digital map overlays of slope, soil, vegetation, drainage, obstacles, transportation (roads, etc.) and the quantity of recent weather (Joe & Feldman, 1998). For example authors (Benton *et al.*, 1995) describe *HERMES* (*Heterogeneous Reasoning and Mediator Environment System*), which allows the answering of queries that require the interrogation of multiple databases in order to determine the start and destination parameters for the route planner.

One of the most popular representations of the terrain is a graph representation. There are a few approaches, in which the map (representing a terrain area) is decomposed into a graph. All of them first convert the map into regions of *go* (*open*) and *no-go* (*closed*). The *no-go* areas may include obstacles and are represented as polygons. A few methods of map representation is used, for example: visibility diagram, Voronoi diagram, straight-line dual of the Voronoi diagram, edge-dual graph, line-thinned skeleton, regular grid of squares, grid of homogeneous squares coded in a quadtree system, etc. (Benton *et al.*, 1995; Schiavone *et al.*, 1995; Schiavone *et al.*, 2000; Tarapata, 2003a).

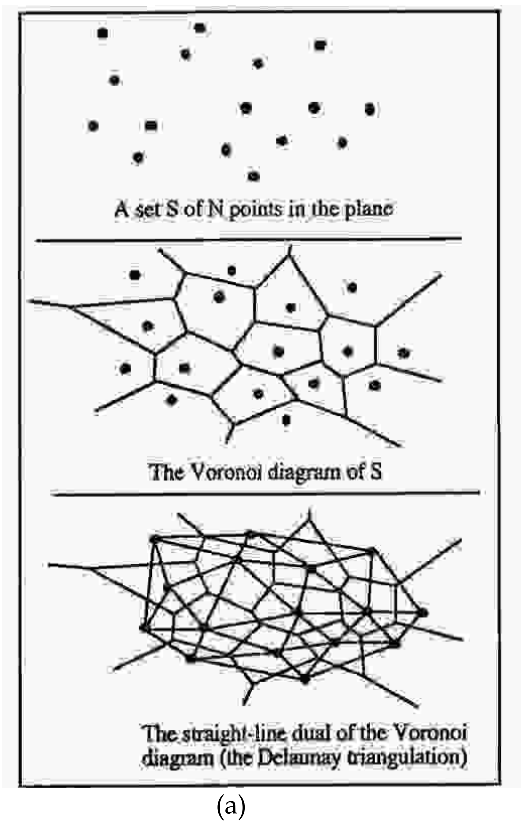
The polygonal representations of the terrain are often created in Database Generated Systems (*DBGS*) through a combination of automated and manual processes (Schiavone *et al.*, 1995; Schiavone *et al.*, 2000). It is important to say that these processes are computationally complicated, but are conducted before simulation (during the preparation process). Typically, an initial polygonal representation is created from the digital terrain elevation data through the use of an automated triangulation algorithm, resulting in what is commonly referred to as a *Triangulated Irregular Network* (*TIN*). A commonly used triangulation algorithm is the Delaunay triangulation. The definition of the Delaunay triangulation may be done via its direct relation to the Voronoi diagram of set S with an N number of 2D points: the straight-line dual of the Voronoi diagram is a triangulation of S.

The *Voronoi diagram* is the solution to the following problem: given set *S* with an *N* number of points in the plane, for each point p_i in *S* what is the locus of points *(x,y)* in the plane that are closer to p_i than to any other point of *S*?

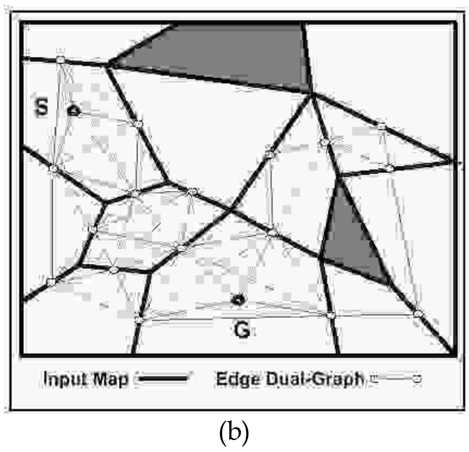
The *straight-line dual* is defined as the graph embedded in the plane obtained by adding a straight-line segment between each pair of points of *S* whose Voronoi polygons share an edge. Fig. 2.1a depicts an irregularly spaced set of points *S*, its Voronoi diagram, and its straight-line dual (i.e. its Delaunay triangulation).

The *edge-dual graph* is essentially an adjacency list representing the spatial structure of the map. To create this graph, we assign a node to the midpoint of each map edge, which does not bound an obstacle (or the border). Special nodes are assigned to the start and goal points. In each non-obstacle region, we add arcs to connect all nodes at the midpoints of the edges, which bound the same region. The fact that all regions are convex, guarantees that all such arcs cannot intersect obstacles or other regions. An example of the edge-dual graph is presented in Fig. 2.1b.

The *visibility graph*, is a graph, which nodes are the vertices of terrain polygons and edges join pairs of nodes, for which the corresponding segment lies inside a polygon. An example is shown in Fig. 2.2. This idea is used to find optimal flight path in a segmented airspace (Kulas *et al.*, 2008).


A set S of N points in the plane
The Voronoi diagram of S
The straight-line dual of the Voronoi diagram (the Delaunay triangulation)

(a)



(b)

Fig. 2.1. (a) Voronoi diagram and its Delaunay triangulation (Schiavone *et al.*, 1995); (b) Edge-dual graph. Obstacles are represented by filled polygons

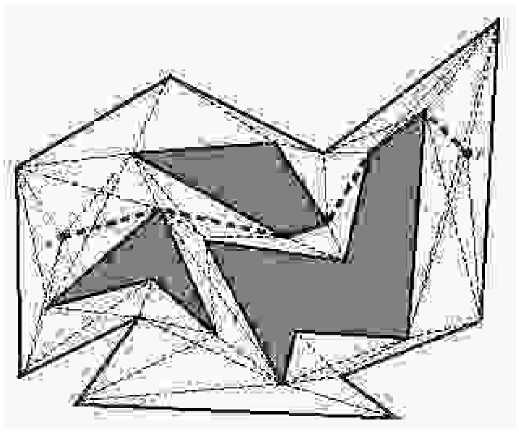


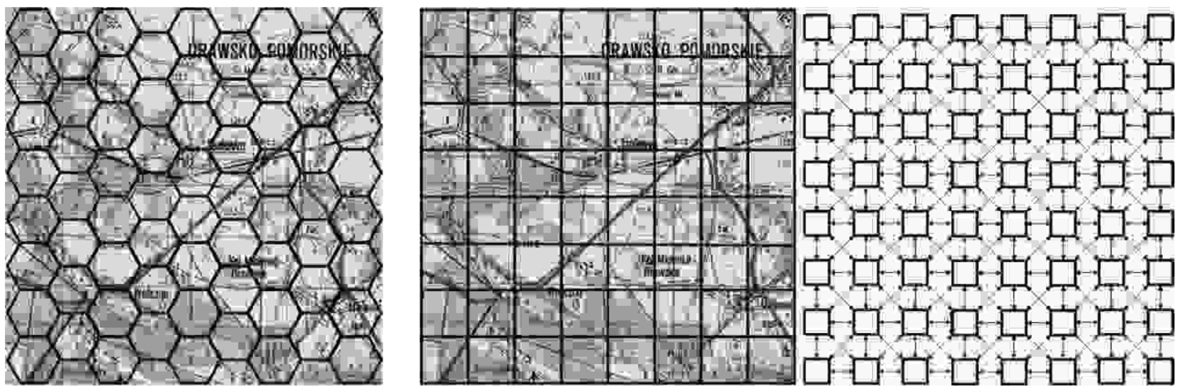
Fig. 2.2. Visibility graph (Mitchell, 1999). The shortest geometric path from the source node *s* to the destination *t* is marked by dashed bold line. Obstacles are represented by filled polygons

The *regular grid (mesh) of squares* (or hexagons, e.g. in *JTLS* system (JTLS, 1988)) divides terrain space into the squares with the same size and each square is treated as having homogeneity from the point of view of terrain characteristics (see Fig. 2.3).

The *grid of homogeneous squares coded in quadtree system* divides terrain space into the squares with a heterogeneous size (Fig. 2.4). The size of the square results from its homogeneity according to terrain characteristics. An example of this approach was presented in (Tarapata, 2000d). This approach represents multiresolution terrain modelling which is also used for battlefield terrain modelling (Behnke, 2004; Cassandras *et al.*, 2000; Chou *et al.*, 1998; Davis *et al.*, 2000; Magillo & Bertocci, 1998; Pai & Reissell, 1994; Tarapata, 2003a). This is a nature of hierarchical structure of military units and methods of their behaviours on a simulated battlefield. For a company level of units greater precision of terrain (environment) model is required than e.g. for the brigade level. Very good definition of multiresolution terrain is presented in (Magillo & Bertocii, 1998):

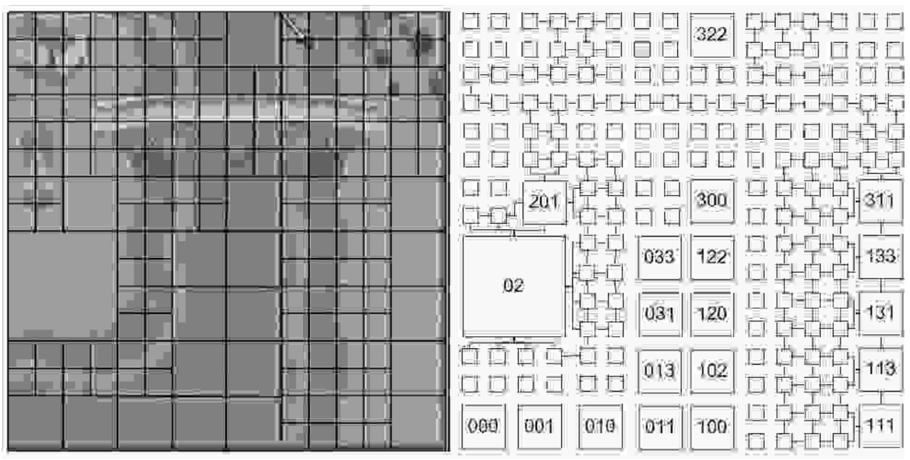
"(...) The concept of multiresolution refers to the possibility of using different representations of a spatial entity, having different levels of accuracy and complexity. Multiresolution models allow trading off accuracy of representation and amount of data to be manipulated. Multiresolution representations of terrains are of great interest when large quantities of data are available and/or large areas are modeled (...)".

In many existing simulation systems there are different solutions regarding terrain representation. In the *JTLS* system (JTLS, 1988) terrain is represented using hexagons with a size ranging from 1km to 16km. In the *CBS* system (CBS, 2001) terrain is similarly represented, but an additional vectoral-region approach is applied. In the *Simulation-Based Operational Training Support System (SBOTTS) Zlocien* (Najgebauer, 2004a; 2004b) and the *System of Automatic Tools for Decision Support (SATDS) – Guru* (Guru, 2005) a dual model of the terrain: (1) as a regular network of terrain squares with square size 200mx200m, (2) as a road-railroad network, which is based on a digital map, is used (Tarapata, 2004b; 2004c). This model is presented in details in chapter 2.3.



(a) (b)

Fig. 2.3. Examples of terrain representation in a simulated battlefield: (a) regular grid of terrain hexagons; (b) regular grid (mesh) of terrain squares and its graph representation with 8 neighbours



(a) (b)

Fig. 2.4. (a) Partitioning of the selected real terrain area into squares of topographical homogeneous areas; (b) Determination of possible links between neighbouring squares and a description of selected vertices in the quadtree system for terrain area presented in (a)

Advantages and disadvantages of terrain representations and their usage for terrain-based movement planning are presented in chapter 2.2.

2.2. Terrain-Based Approaches for Paths Planning

There are four main approaches concerning terrain representation that are used in a battlefield decision support and simulation for paths planning (Karr *et al.*, 1995): free space analysis, vertex graph analysis, potential fields and grid-based algorithms.

In the *free space approach*, only the space not blocked and occupied by obstacles is represented. For example, representing the centre of movement corridors with Voronoi diagrams (Schiavone *et al.*, 1995) is a free space approach (see Fig. 2.1). The advantage of Voronoi diagrams is that they have efficient representation. Disadvantages of Voronoi diagrams are as follows: they tend to generate unrealistic paths (paths derived from Voronoi diagrams follow the centre of corridors while paths derived from visibility graphs clip the edges of obstacles); the width and trafficability of corridors are typically ignored; distance is generally the only factor considered in choosing the optimal path.

In the *vertex graph approach*, only the endpoints (vertices) of possible path segments are represented (Mitchell, 1999). Advantages of this approach: it is suitable for spaces that have sufficient obstacles to determine the endpoints. Disadvantages are as follows: determining the vertices in "open" terrain is difficult; trafficability over the path segment is not represented; factors other than distance can not be included in evaluating possible routes.

In the *potential field approach*, the goal (destination) is represented as an "attractor", obstacles are represented by "repellers", and the vehicles are pulled toward the goal while being repelled from the obstacles. Disadvantages of this approach: the vehicles can be attracted into box canyons from which they can not escape; some elements of the terrain may simultaneously attract and repel.

In the *regular grid approach*, the grid overlays the terrain, terrain features are abstracted into the grid, and the grid rather than the terrain is analyzed. Advantages are as follows: analysis simplification. Disadvantages: "jagged" paths are produced because movement out of a grid cell is restricted to four (or eight) directions corresponding to the four (or eight) neighbouring cells; granularity (size of the grid cells) determines the accuracy of terrain representation.

In many of the existing simulation systems there are different solutions regarding this subject (Benton *et al.*, 1995; Campbell *et al.*, 1995, Kreitzberg *et al.*, 1990; Longtin & Megherbi, 1995; Tarapata, 2003a). In the work of (Benton *et al.*, 1995) authors describe a combined on-road/off-road planning system that was closely integrated with a geographic information system and a simulation system. Routes can be planned for either single columns or multiple columns. For multiple columns, the planner keeps track of the temporal location of each column and insures they will not occupy the same space at the same time. In the same paper the *Hierarchic Route Planner* as the integrate part of the *Predictive Intelligence Military Tactical Analysis System (PIMTAS)* is discussed. In the paper (James *et al.*, 1999) authors presented an on-going effort to develop a prototype for ground operations planning, the *Route Planning Uncertainty Manager (RPLUM)* tool kit. They apply uncertainty management to terrain analysis and route planning since this activity supports the commander's scheme of manoeuvre from the highest command level down to the level of each combat vehicle in every subordinate command. They extend the *PIMTAS* (Benton *et al.,* 1995) route planning software to accommodate results of reasoning about multiple categories of uncertainty. Authors of the paper (Campbell *et al.,* 1995) presented route planning in the *Close Combat Tactical Trainer (CCTT).* Kreitzberg (Kreitzberg *et al.*, 1990) has developed the *Tactical Movement Analyzer (TMA).* The system uses a combination of digitized maps, satellite images, vehicle type and weather data to compute the traversal time across a grid cell. *TMA* can compute optimum paths that combine both on-road and off-road mobility, and with weather conditions used to modify the grid cost factors. The smallest grid size used is approximately 0.5 km. The author uses the concept of a signal propagating from the starting point and uses the traversal time at each cell in the array to determine the time at which the signal arrives at neighbouring cells. A lot of these systems use the *Continuous Dijkstra's Algorithm* for route planning described by Mitchell in (Mitchell, 1999). In the simulation-based operational training support system *SBOTSS Zlocien* (Najgebauer, 2004a) a dual model of the terrain: (1) as regular network of terrain squares with a square size of 200mx200m, (2) as road-railroad network, which is based on a digital map in VPF format, is used. To find paths for units, modified shortest path algorithms (*SPA*) such as Dijkstra's, A*, geometric *SPA* are used. Geometric *SPA* supplements two algorithms presented above (the hybrid shortest path algorithm is obtained) and it is used in case the size of the network is large (default is 10000 nodes, but it is a parameter set in a so-called calibrator of the simulation system (Antkiewicz *et al.*, 2006)). Modifications of mentioned algorithms deal with the following details: (a) paths determination in different configurations - (a1) from point (region) to point (region), (a2) visiting selected points (regions), (a3) omitting selected points (regions, obstacles), (a4) inside or outside a selected region, (a5) off-roads only, (a6) on-roads only, (a7) combined on- and off-roads and others; (b) if we do not set the region inside where we want to find the path then the algorithm itself, iteratively determines the rectangular region, which is based on a line linking the beginning and end points (nodes) of movement, to minimize computational time; (c) if we want to find an on-road path only, and there are no nodes of the road network inside the intermediate squares, then the algorithm may optionally find crossroads (nodes of the road network), which are nearest to squares inside that the path must cross. Detailed description of the movement planning algorithms used in *SBOTSS Zlocien* is presented in (Tarapata, 2004b; 2004c). Moreover, it is also presented in chapter 6.1. A special type of system for movement planning is *Allied Deployment and Movement System* (*ADAMS*) (Heal & Garnett, 2001), which has been developed in support of multinational force movement planning. This system is in wide use throughout NATO and nations for analysis, generation and coordination of movement plans. The *ADAMS* provides the users with the tools to plan and manage deployment operations.

Taking into account multiresolution terrain modelling this approach is also used for battlefield modelling and simulation. For example, in the papers (Tarapata, 2004b; 2010a; 2010c) a decomposition method, and its properties, which decreases computational time for path searching in multiresolution graphs has been presented. The goal of the method is not only computation time reduction but, first of all, using it for multiresolution path planning (to apply similarity in decision processes on a different command level and decomposing-merging approach). The method differs from very effective representations of terrain using quadtree (Kambhampati & Davis, 1986) because of two main reasons: (1) elements of quadtree, which represent a terrain have irregular sizes, (2) in a majority of applications quadtree represents only binary terrain with two types of region: open (passable) and closed (impassable). Hence, this approach is very effective for mobile robots, but it is not adequate, for example, to represent the battlefield environment (Tarapata, 2003a).

Some models and algorithms for terrain-based movement planning are considered in detail in chapter 3.

2.3. Terrain Model in the *Zlocien* System as an Example of Battlefield Environment Model

The terrain (environment) model S_0, which we use as a battlefield model in the *Zlocien* system (Najgebauer, 2004a, 2004b) is based on the digital map in *VPF* format. The model is twofold: (1) as a regular network Z_1 of terrain squares, (2) as a road-railroad network Z_2 and it is defined as follows (Tarapata, 2004c; 2004d):

$$S_O(t) = \langle Z_1(t), Z_2(t) \rangle \tag{2.1}$$

The regular grid of squares Z_1 (see Fig. 2.3b) divides terrain space into squares with the same size (200m×200m) and each square is homogeneous from the point of view of terrain characteristics (degree of slowing down velocity, ability to camouflage, degree of visibility, etc.). This square size results from the fact that the lowest level of modelled units in *SBOTSS Zlocien* is a platoon and 200m it is approximately the width of the platoon front during attack. The Z_1 model is used to plan off-road (cross-country) movement e.g. during attack planning. In the Z_2 road-railroad network (see Fig. 2.5) we have crossroads as network nodes and section of the roads linking adjacent crossroads as network links (arcs, edges). This model is used to plan fast on-road movement, e.g. during march (redeployment) planning and simulation. Movement planning and simulation methods in *Zlocien* system using Z_1 and Z_2 models are described in chapter 6.1.

Models Z_1 and Z_2 are integrated. This integration gives possibilities to plan movement taking into account both models. It is possible, because each square of terrain contains information about fragments of road inside this square. On the other hand each fragment of road contains information on squares of terrain, which they cross. Hence, the route for any object (unit) may consist of sections of roads and squares of terrain. It is possible to get off the road (if it is impassable) and start off-road movement (e.g. omit impassable section of road) and afterwards returning to the road. Conversely, we can move off-roads (e.g. during attack), access a section of road (e.g. any bridge to go across the river) and then return back off-road (on the other riverside). The characteristics of both terrain models depend on: time, terrain surface and vegetation, weather, the time of day and season of the year, opponent and own destructions (e.g. destruction of the bridge, which is element of road-railroad network) (see Table 2.1 and Table 2.2).

The formal definition of the regular network of terrain squares Z_1 is as follows (see Fig. 2.3b):

$$Z_1(t) = \langle G_1, \Psi_1(t) \rangle \tag{2.2}$$

where G_1 defines Berge's graph[1] describing the squares network structure, $G_1 = \langle W_1, \Gamma_1 \rangle$, W_1 - set of graph's nodes (terrain squares); $\Gamma_1 : W_1 \to 2^{W_1}$ - function describing for each nodes of the G set of adjacent (direct successors) nodes (maximal 8 adjacent nodes); $\Psi_1(t) = \{\Psi_{1,0}(\cdot,t), \Psi_{1,1}(\cdot,t), \Psi_{1,2}(\cdot,t), ..., \Psi_{1,LW_1}(\cdot,t)\}$ - set of functions defined on the graph's nodes (depending on t).

One of the functions of $\Psi_1(t)$ is the function of slowing down velocity $FSDV(n,\ldots)$, $n \in W_1$, which describes slowing down velocity (as a real number from [0,1]) inside the n-th square of the terrain,

$$FSDV : W_1 \times T \times Veh \times Meteo \times YearS \times DayS \to [0,1] \tag{2.3}$$

where: T – set of times, Veh – set of vehicle types, Veh ={Veh_Wheeled, Veh_Wheeled-Caterpillar, Veh_Caterpillar}; $Meteo$ – set of meteorological conditions, $YearS$ – set of the seasons of year, $DayS$ – set of the times of day.

The function $FSDV$ is used to calculate crossing time between two squares of terrain. Other functions (as subset of $\Psi_1(t)$) described on the nodes (squares) of G_1 and essential from the point of view of trafficability and movement are presented in Table 2.1.

[1] Berge's graph is such a directed graph which has at most one arc between each ordered pairs of nodes. One of the formal definitions is presented in (Korzan, 1978).

Table 2.1. The most important functions described on the terrain square (node of G_1)

Description of the function	*Definition of the function*
Geographical coordinates of node (centre of square)	$FWSP: W_1 \to R^3$
Ability to camouflage in the square	$FCam: W_1 \times T \to [0,1]$
Degree of terrain undulation in the square	$FUnd: W_1 \to [0,1]$
Subset of the node's set of Z_2 network, which are located inside the square	$FW_1OnW_2: W_1 \to 2^{W_2}$

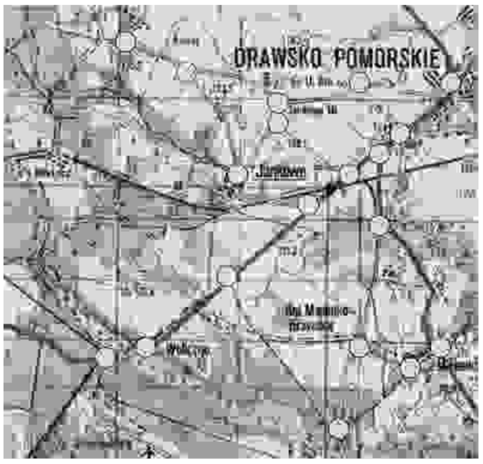
The formal definition of the road-railroad network Z_2 is following (see Fig. 2.5):

$$Z_2(t) = \langle G_2, \Psi_2(t), \zeta_2(t) \rangle \tag{2.4}$$

where G_2 describes Berge's graph defining structure of road-railroad network, $G_2 = \langle W_2, U_2 \rangle$, W_2 - set of graph's nodes (crossroads); $U_2 \subset W_2 \times W_2$ - set of graph G_2 arcs (sections of roads); $\Psi_2(t) = \{\Psi_{2,0}(\cdot,t), \Psi_{2,1}(\cdot,t), ..., \Psi_{2,LW_2}(\cdot,t)\}$ - set of functions defined on the graph's G_2 nodes (depending on t); $\zeta_2(t) = \{\zeta_{2,i}(\cdot,t)\}_{i=\overline{1,IG_2}}$ - set of functions defined on the graph's G_2 arcs (depending on t).

Functions (as subset of $\Psi_2(t)$ and $\zeta_2(t)$) are presented, which are essential from the point of view of trafficability and movement, described on the nodes and arcs of G_2 in Table 2.2. One of the most important functions is slowing down velocity function $FSDV2(u,...)$, $u \in U_2$ which describes slowing down velocity (as real number from [0,1]) on the u-th arc (section of road) of the graph:

$$FSDV2: U_2 \times T \times Veh \times Meteo \times YearS \times DayS \to [0,1] \tag{2.5}$$



(a)

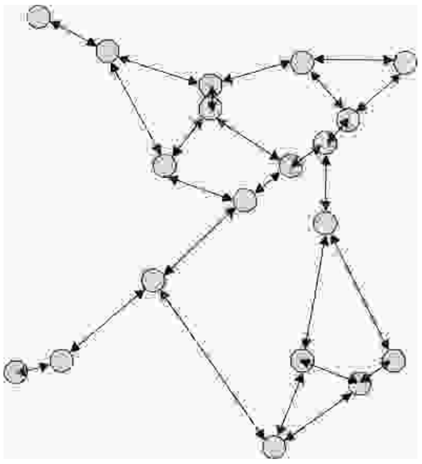

(b)

Fig. 2.5. Road-railroad network Z_2 (a) and its graph model G_2 (b)

Table 2.2. The most important functions described on the crossroads and on part of the roads (G_2)

Description of the function	*Definition of the function*
Geographical coordinates of node (crossroad)	$FWSP2 : W_2 \to R^3$
Node from Z_1, which contains node from Z_2	$FW_2OnW_1 : W_2 \to W_1$
Subset of set of the nodes of Z_1 network, which contains the arc	$FU_2OnW_1 : U_2 \to 2^{W_1}$
Degree of terrain undulation on the arc	$FUnd : U_2 \to [0,1]$
Arc length	$FLen : U_2 \to R^+$

For movement planning models in *Zlocien* system, described in chapter 6.1 and in (Tarapata, 2004b; 2004c), we construct some temporary network S^z:

$$S^z = \left\langle G^z, \Psi_1(t) \cup \Psi_2(t), \zeta_2(t) \cup \{l_1, l_2, l_3\} \right\rangle \tag{2.6}$$

where: G^z – Berge's graph describing structure of the temporary network (Fig. 2.6),

$$G^z = \left\langle W^z, U^z \right\rangle \tag{2.7}$$

$W^z = W_1 \cup W_2$ – set of graph's G^z nodes, W_1 defined in (2.2), W_2 defined in (2.4); $U^z \subset W^z \times W^z = U_1 \cup U_2 \cup U_3$ – set of graph's G^z arcs, U_2 described in (2.4) and

$$U_1 = \{(a,b) \in W_1 \times W_1 : b \in \Gamma_1(a)\} \tag{2.8}$$

$$U_3 = U_3^{'} \cup U_3^{''} \tag{2.9}$$

$$U_3^{'} = \{(a,b) \in W_1 \times W_2 : FW_2OnW_1(b) \in \Gamma_1(a)\} \tag{2.10}$$

$$U_3^{''} = \{(a,b) \in W_2 \times W_1 : b \in \Gamma_1(FW_2OnW_1(a))\} \tag{2.11}$$

l_1 – function which describes crossing time by an arc:

$$l_1 : U^z \to R^+ \cup \{0\} \tag{2.12}$$

l_2 – function describing geometrical length of an arc:

$$l_2 : U^z \to R^+ \tag{2.13}$$

l_3 – function describing ability to camouflage on an arc:

$$l_3 : U^z \to [0,1] \tag{2.14}$$

Let's note that we determine values of l_1, l_2 and l_3 in the moment T_0, in which we plan the movement for each arc $(a,b) \in U^z$. Therefore, they depend on time but we omit it to simplify descriptions. Moreover, we accept following notations: $met(T_0) \in Meteo$ – meteorological conditions on the arc (a,b) in the moment T_0; $pr(T_0) \in YearS$ – the season of the year inside the region in the moment T_0; $pd(T_0) \in DayS$ – the time of the day inside the region in the moment T_0;
$veh(p) \in Veh$ – type of the vehicle p.

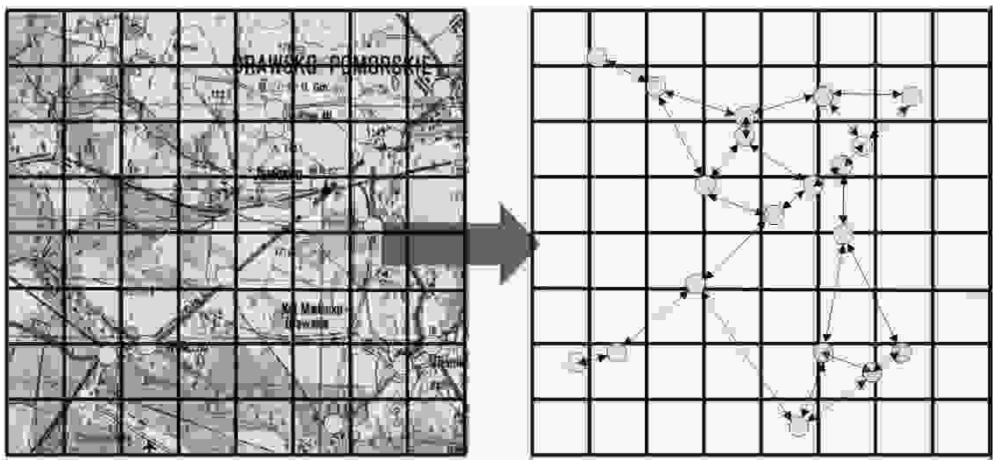


Fig. 2.6. Structure G^z of temporary network S^z

We define l_1 function as follows:

$$l_1((a,b)) = \begin{cases} \dfrac{d(a,b)}{v^{slowd}(id,(a,b))}, & \text{when } v^{slowd}(id,(a,b)) \neq 0 \\ \infty, & \text{otherwise} \end{cases} \tag{2.15}$$

where: $d(a,b)$ – geometric distance between nodes a, b,

$$d(a,b) = \sqrt[3]{(x(a)-x(b))^3 + (y(a)-y(b))^3 + (z(a)-z(b))^3} \tag{2.16}$$

$x(w)$, $y(w)$, $z(w)$ – describe coordinates of node w (calculated using functions *FWSP* (see Table 2.1) when $w \in W_1$ or *FWSP2* (see Table 2.2) when $w \in W_2$),
$v^{slowd}(id,(a,b))$ – maximal velocity of the unit *id* on the arc (a,b) taking into account topographical conditions,

$$v^{slowd}(id,(a,b)) = v^{\max}(id) \cdot FOP(id,(a,b)) \tag{2.17}$$

$v^{\max}(id)$ – maximal possible velocity of the unit *id* resulting from technical parameters of the vehicles belonging to this unit,

$$v^{\max}(id) = \min_{p \in ZVeh(id)} v^{tech}(p) \tag{2.18}$$

$ZVeh(id)$ – set of vehicles belonging to the *id* unit, $v^{tech}(p)$ – maximal velocity of the vehicle p (resulting from its technical parameters),
$FOP(id,(a,b))$ – slowing down velocity function for the *id* unit on the arc (a,b),

$$FOP(id,(a,b)) =$$

$$= \begin{cases} \dfrac{\sum\limits_{p\in ZVeh(id)} FSDV2((a,b),T_0,veh(p),met(T_0),pr(T_0),pd(T_0))}{\overline{\overline{ZVeh(id)}}}, \text{ when } a,b\in W_2 \\ \dfrac{\sum\limits_{p\in ZVeh(id)} FSDV(a,\square,veh(p),\ldots)+\sum\limits_{p\in ZPoj(id)} FSDV(b,\square,veh(p),\ldots)}{2\overline{\overline{ZVeh(id)}}}, \text{ when } a,b\in W_1 \\ \dfrac{\sum\limits_{p\in ZVeh(id)} FSDV(a,\square,veh(p),\ldots)}{2\overline{\overline{ZVeh(id)}}}+ \\ +\dfrac{\sum\limits_{p\in ZVeh(id)} FSDV(FW_2OnW_1(b),\square,veh(p),\square,\square,\square)}{2\overline{\overline{ZVeh(id)}}}, \text{ when } a\in W_1, b\in W_2 \\ \dfrac{\sum\limits_{p\in ZVeh(id)} FSDV(FW_2OnW_1(a),\square,veh(p),\square,\square,\square)}{2\overline{\overline{ZVeh(id)}}}+ \\ +\dfrac{\sum\limits_{p\in ZVeh(id)} FSDV(b,\square,veh(p),\square,\square,\square)}{2\overline{\overline{ZVeh(id)}}}, \text{ when } a\in W_2, b\in W_1 \end{cases} \tag{2.19}$$

Function l_2 is defined as follows:

$$l_2((a,b)) = \begin{cases} d(a,b), & \text{when } (a,b)\in U^z \\ \infty, & \text{otherwise} \end{cases} \tag{2.20}$$

Function l_3 is defined as follows:

$$l_3((a,b)) = \begin{cases} \dfrac{FCam(a,T_0)+FCam(b,T_0)}{2}, & \text{when } a,b\in W_1 \\ \dfrac{FCam(FW_2OnW_1(a),T_0)}{2}+\dfrac{FCam(b,T_0)}{2}, & \text{when } a\in W_2, b\in W_1 \\ \dfrac{FCam(a,T_0)}{2}+\dfrac{FCam(FW_2OnW(b),T_0)}{2}, & \text{when } a\in W_1, b\in W_2 \\ \dfrac{FCam(FW_2OnW(a),T_0)}{2}+\dfrac{FCam(FW_2OnW(b),T_0)}{2}, & \text{when } a,b\in W_2 \end{cases} \tag{2.21}$$

We use these functions in chapters 5.3 and 6.1. Similar terrain model is used in the *SATDS – Guru* (Antkiewicz *et al.*, 2009c; Najgebauer, 2008a).

In the *Zlocien* system some terrain classification method (Najgebauer & Tarapata, 2004d) for decision automata for an attack and defence on the tactical level which is based on the defined terrain model is also used. This method is one of the part of the first stage of automata described in (Antkiewicz *et al.*, 2003; 2004a; 2004b; Najgebauer *et al.*, 2007b) and in chapter 5.2, and it is based on presented model of the terrain. The idea of the method is to estimate terrain region in which own and opposite units will operate to obtain one of the four of kinds of the terrain: *go, slow go, no go, no move*. The first kind of the terrain (*go*) is excellent for movement (e.g. plain terrain), the second one (*slow go*) is good for movement (e.g. soft-hilly terrain), the third kind of the terrain (*no go*) is poor for movement (e.g. hard-hilly terrain or mountainous terrain) and the last kind of the terrain (*no move*) describes impassable terrain (e.g. lakes, seas, high mountains). The region (action strip) in which own and opposite units will operate is divided into rectangular or trapezoidal subregions (each of these for subordinate unit). Inside each of the subregions and between adjacent subregions we determine shortest paths from the start to the end of the region (the start of the region is taking from the side of own units and the end of the region is taking from the side of opposite units). These paths are determined taking into considerations all characteristics having influence on movement in the subregions and between adjacent subregions (terrain topography, weather, the time of the day, season of the year). The movement planning algorithms use modifications of shortest paths algorithms (*SPA*) such as: Dijkstra's *SPA*, Johnson's *SPA* in thin networks, *A* SPA*, geometric *SPA* (see chapter 6.1.2). After this step we obtain square matrix with dimensions: *number_of_subregions*×*number_of_subregions* which elements $s_{k,l,j} \in [0,1]$ equals relation between time on the shortest path from start of the region *l* to end of the region *j* and between minimal travel time from start of the region *l* to end of the region *j* inside the subregion *k* under ideal environmental conditions. Estimation $\overline{S}_k$ of the *k*-th region equals mean value from among $s_{k,l,j}$. The region of the terrain is classified as *go, slow go, no go, no move* if estimation $\overline{S}_k$ of the region is not greater than some critical value (set as parameters of simulator to calibrate terrain classification, (Antkiewicz *et al*., 2006)). The kind of the terrain determined using described method is component of classification vector which define the decision situation in automata (Antkiewicz *et al*., 2003; 2004a; 2004b). On the basis of this vector the variants of decisions are generated and the optimal decision is selected.

3. Models and Algorithms for Movement Planning

3.1. Introduction

Movement (paths, routing, motion) planning is an essential element in many applications (LaValle, 2006): transportation, computer networks, mobile robots, car navigation, virtual agents in computer games, etc. From the point of view of military application, explained in this monograph, it is very interesting. Object movement is an essential element of combat actions and it is related to manoeuvre planning of military detachments on the battlefield during battle as well as preparing for it. This process is very important from the point of view of simulating complex processes in military systems. It may have an effect on accuracy, adequateness, effectiveness and other characteristics of these systems. Redeployment planning and simulation of military objects is a basic problem especially in combat simulators or *CGFs*. As an inseparable part of *CGF*, modules for route planning based on the real-terrain models are used. They have submodules to generate digital terrain and for route planning they use processed terrain information. For example, in *ModSAF (Modular Semi-Automated Forces)* in module "SAFsim", which simulates the entities, units, and environmental processes the route planning component is located (Longtin & Megherbi, 1995). Other terrain-based path planning modules have been described in chapter 2.2.

Many route planners in the literature are based on the *off-line path planning algorithms*: a path for the object is determined before its movement. These algorithms are divided into two groups (Zhan & Noon, 2000): *label setting algorithms* and *label correcting algorithms*. The following are exemplary algorithms of the *label setting* approach: modified Dijkstra's algorithm (Dijkstra, 1959) with a priority queue represented by *d*-ary heap ($O(A\log_d V)$, where V – number of nodes of a graph, A – number of edges (or arcs) of a graph, $d = \max\{2, \lceil A/V \rceil\}$) proposed in (Tarjan, 1983), with priority queue represented by Fibonacci heap ($O(A+V\log V)$) proposed in (Fredman & Tarjan, 1987), with buckets (Zhan & Noon, 1998), symmetric Dijkstra's algorithm (Zhao, 1997), A* algorithm (average time proportional to $O(\sqrt{V}\cdot V)$) (Korf, 1999). Very interesting group are geometric path planning algorithms (Mitchell, 1999) or its variants (Korf, 1999; Logan, 1997a; Logan & Sloman, 1997b; Rajput & Karr, 1994; Tarapata, 1999a; 2001; 2003a; 2004a; Undeger *et al.*, 2001). As *label correcting algorithms* we can apply: Bellmann-Ford's algorithm with complexity $O(VA)$, Pallottino algorithm (Pallottino & Scutella, 1998), *PDM* algorithm or others (Gabow-Tarjan's algorithm (Gabow & Tarjan, 1989) with complexity $O(\sqrt{V}A\log(VW)$ where W is the largest absolute weight of edges) or the algorithm presented in (Ahuja *et al.*, 1988) ($O(A+V\sqrt{\log W}$))).

For finding all-pairs shortest paths we can apply V times (for each node) the modified Dijkstra's algorithm ($O(VA\log_d V)$), Johnson's algorithm in sparse networks (Johnson, 1977) ($O(V^2 log V+VA)$) or algorithms in *DAGs* (directed acyclic graphs) e.g. the Bellman algorithm ($O(V+A)$). For example, A^* has been used in a number of *Computer Generated Forces* systems as the basis of their component planning, to plan road routes (Campbell *et al.*, 1995), to avoid moving obstacles (Karr *et al.*, 1995), to avoid static obstacles (Rajput & Karr, 1994) and to plan concealed routes (Longtin & Megherbi, 1995). Moreover, the multicriteria approach to the path determined in *CGF* systems is often used. Some results of selected multicriteria paths problem and analysis of the possibility to use them in *CGF* systems are described, e.g. in (Tarapata, 2007d). A very extensive discussion related to geometric shortest path planning algorithms was presented by Mitchell in (Mitchell, 1999) (references consist of 393 papers and handbooks). The geometric shortest path problem is defined as follows: given a collection of obstacles, find an Euclidean shortest obstacle-avoiding path between two given points. Mitchell considers the following problems: geodesic paths in a simple polygon; paths in a polygonal domain (searching the visibility graph, continuous Dijkstra's algorithm); shortest paths in other metrics (L_p metric, link distance, weighted region metric, minimum-time paths, curvature-constrained shortest paths, optimal motion of non-point robots, multiple criteria optimal paths, sailor's problem, maximum concealment path problem, minimum total turn problem, fuel-consuming problem, shortest paths problem in an arrangement); on-line algorithms and navigation without map; shortest paths in higher dimensions.

The basic idea of the *on-line path planning algorithms* (Korf, 1999), in general, is that the object is moved step-by-step from cell to cell using a heuristic method. This approach is borrowed from robots motion planning (Behnke, 2004; Kambhampati & Davis, 1986; LaValle, 2006; Logan & Sloman, 1997; Undeger *et al.*, 2001). The decision about the next move (its direction, speed, etc.) depends on the current location of the object and environment status. Examples of on-line path planning algorithms (Korf, 1999): *RTA** (*Real-Time A**), *LRTA** (*Learning RTA**), *RTEF* (*Real-Time Edge Follows*), *HLRTA**, *eFALCONS*. For example, the idea of *RTEF* algorithm (Undeger *et al.*, 2001) is to let the object eliminate closed directions (the directions that cannot reach the target point) in order to decide on which way to go (open directions). For instance, if the object has a chance to realize that moving north and east will not let him reach the goal state, then it will prefer to go south or west. *RTEF* finds out these open and closed directions by decreasing the number of choices the object has.

However, the on-line path planning approach has one basic disadvantage: in this approach using a few criterions simultaneously to find an optimal (or acceptable) path is difficult and it is rather impossible to estimate, the moment of reaching the destination in advance. Moreover, it does not guarantee finding optimal solutions and even suboptimal ones may significantly differ from acceptable solutions.

Organization of this chapter is as follows: chapter 3.2 contains decomposition and a multiresolution approach to path planning (based on the papers (Tarapata, 2004a; 2010a; 2010c)), in chapter 3.3 models and algorithms for multiobjective (multicriteria) paths planning have been described (based on the papers (Tarapata 1999a; 2000e; 2005c; 2007d)), chapter 3.4 contains analysis of specific disjoint paths planning models and algorithms (based on the papers (Tarapata 2006b; 2008e; 2010g; 2011d)). Presented applications and examples of methods being described concern military applications but these methods are interdisciplinary.

3.2. Decomposition and Multiresolution Paths Planning

3.2.1. Description of the Problem

Multiresolution paths (paths in multiresoultion environment model, see chapter 2.1) are very interesting from many applications point of view (mobile robots (Ahuja *et al.*, 1988; Kambhampati & Davis, 1986; LaValle, 2006), battlefield simulation (Tarapata, 2003a), *Computer Generated Forces* (Petty, 1995), transportation or navigation (Chou *et al.*, 1998). These are fields, which describe either the size of the environment or environment complexity (3D terrain). For example, in a battlefield decision support and simulation systems, planning models of movement based on a multiresolution environment (see definition in chapter 2.1) are used. This is the nature of a hierarchical structure of military units and methods of their behaviours on a simulated battlefield. For a company level of units, greater precision of terrain (environment) model is required than, for example, the brigade level (see details in chapter 3.2.6).

The multiresolution paths problem is strongly connected with the problem of finding the shortest paths in large-scale networks. There are two main approaches to the shortest paths problem in large-scale networks: (a) to decompose a problem or environment (network, graph) in which we plan into smaller problems and then solve subproblems (Ahuja *et al.*, 1988; Kambhampati & Davis, 1986; Pai & Reissell, 1998); (b) to apply on-line algorithms which find and "merge" path cell-by-cell (Didjev *et al.*, 1995; Korf, 1999; Tarapata, 2003a). The first group of approaches is called multiresolution methods. As local algorithms inside all of these methods, algorithms described in chapter 3.2.1 are used. For example, authors of (Kambhampati & Davis, 1986) present a method based on cell decomposition and partitioning space into a quadtree and then use a staged search (similar to A* algorithm) to exploit the hierarchy. The goal of the approaches presented in (Pai & Reissell, 1998) is to navigate a robot without violating terrain dependent constraints decomposing the terrain with wavelet decomposition. Authors of the paper (Chou *et al.*, 1998) present some *Hierarchical Algorithm* (*HA*), which is designed to look for paths in large networks representing road networks.

Subchapter 3.2.3 presents a decomposition method (*DSP* – decomposition shortest paths) and its properties, which decrease computational time of path searching in multiresolution and large graphs. The goal of the method is not only computation time reduction but, most of all, using it for multiresolution path planning. Presented in chapter 3.2.6 is the method of how to use it for multiresolution battlefield modelling and paths planning.

3.2.2. Definitions and Notations

Let graph $G=\langle V_G, A_G\rangle$ be given (see Fig. 3.1b) as a representation of an example of terrain squares (see Fig. 3.1a), where V_G describes a set of nodes (squares of terrain), $V=\overline{\overline{V_G}}$, A_G describes a set of arcs,

$$A_G \subset \{\langle x,y\rangle \subset V_G \times V_G : \text{ square } x \text{ is adjacent to square } y\},\ A=\overline{\overline{A_G}}.$$

For each arc $\langle x,y\rangle \in A_G$ we have cost $c(x,y)$ value as the crossing time ($c(x,x)=0$, $c(x,y)=+\infty$ when $\langle x,y\rangle \notin A_G$). The problem is to find the shortest path from the source node s to the destination node t in G with the assumption that G is large in size and, simultaneously, to prepare the data structure (a graph) for multiresolution path planning. The idea of the approach is to merge geographically adjacent small squares (nodes belonging to V) into bigger squares (called b-nodes, see Fig. 3.1c) and to build b-graph G^* (graph based on the b-nodes, see Fig. 3.1d) using a specific transformation. This transformation is based on the assumption that we set an arc (b-arc) between two b-nodes $x^* \subset V_G, y^* \subset V_G$ when two such nodes as $x \in x^*, y \in y^*$ exist and that $\langle x,y\rangle \in A_G$ (x and y are called "border" nodes).

Formal definition of the graph G^* is as follows: $G^*=\langle V_G^*, A_G^*\rangle$, $V_G^*=\{x_1^*, x_2^*, ..., x_n^*\}$ – set of b-nodes, $\overline{\overline{V_G^*}}=n$, $\overline{\overline{A_G^*}}=m$, $x_i^*=\{x_{i1}, x_{i2}, ..., x_{im_i}\} \subset V_G$ and each x_i^*, $i=\overline{1,n}$ generates subgraph of G,

$$A_G^*=\left\{\langle x^*, y^*\rangle \subset V_G^* \times V_G^* : \underset{x\in x^*, y\in y^*}{\exists} \langle x,y\rangle \in A_G\right\} \tag{3.1}$$

Let us note that parameter dn (length of the b-node side, see Fig. 3.1c) may be used instead of parameter n for creating graph G^* and it may be computed as follows: $dn=\sqrt{V/n}$ when $\left(V \bmod dn^2\right)=0$.

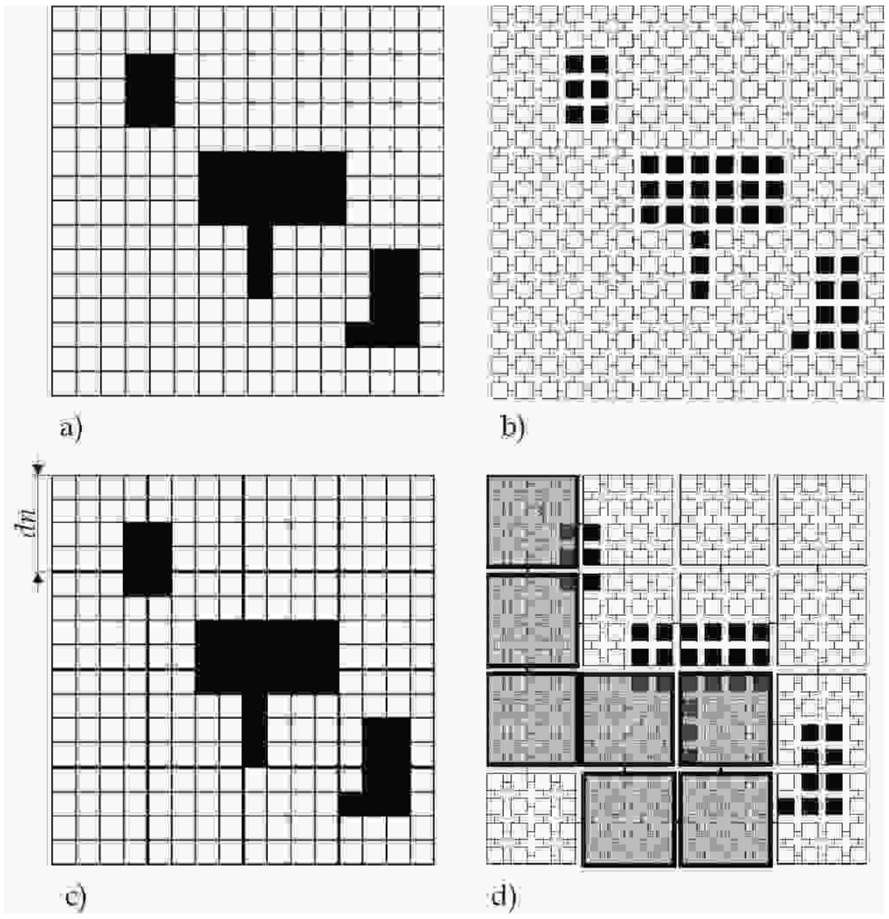


Fig. 3.1. Principles of G^* creating. Regions (squares) with black colour are impassable. a) Terrain space divided into a regular-size grid; b) Grid graph as a representation of terrain squares from a), only north-east-south-west arcs are permitted; c) Merging geographically adjacent small squares from b) into n=16 b-nodes (big squares); d) b-graph G^* for squares merging from c) with marked shortest s-t path in G^* (an edge represent two arcs with opposite directions)

The cost of the b-arc $\langle x^*, y^* \rangle \in A^*_G$ is set as $c^{*\min}(x^*, y^*)$ and $c^{*\max}(x^*, y^*)$: $c^{*\min}(x^*, y^*)$ is represented by the cost vector of the shortest of the shortest paths from any node belonging to x^* to any node belonging to y^* for each predecessor z^* of x^*. This is a vector, because the cost from x^* to y^* depends on the node, from which we achieve x^* (therefore, for each predecessor of x^* we have a cost value, see Fig. 3.2). This cost is calculated inside the subgraph built on the nodes belonging to x^*, y^* and z^*. Cost $c^{*\max}(x^*, y^*)$ is represented by the cost vector of the longest of the shortest paths from any node belonging to x^* to any node belonging to y^* for each predecessor z^* of x^*.

For further discussion we will use the following notations:

$W(x^*, y^*)$ – subset of nodes belonging to x^*, which are adjacent ("border") to any node of y^*, $W(x^*, y^*) = \left\{ x \in x^* : \underset{y \in y^*}{\exists} \langle x, y \rangle \in A_G \right\}$,

$D(x,y)$ – set of paths between nodes x and y in graph G;

$d(x,y)=(x_0=x,x_1,...,x_{l(d(x,y))}=y)$ – element of $D(x,y)$, $\underset{i=\overline{0,l(d(x,y))-1}}{\forall}\langle x_i,x_{i+1}\rangle\in A_G$,

$L(d(x,y))=\sum_{i=0}^{l(d(x,y))-1}c(x_i,x_{i+1})$ – cost of path $d(x,y)$ from x to y;

$D^{\min}(W(x^*,z^*),W(y^*,v^*))$ – set of shortest paths in G between nodes belonging to $W(x^*,z^*)$ and $W(y^*,v^*)$:

$$D^{\min}\left(W(x^*,z^*),W(y^*,v^*)\right)=\left\{\begin{array}{r}d^{\min}(x,y)\in D(x,y): x\in W(x^*,z^*), y\in W(y^*,v^*),\\ L(d^{\min}(x,y))=\min\limits_{d(x,y)\in D(x,y)}L(d(x,y))\end{array}\right\}$$

$c^{*\min}(x^*,y^*)$ – minimal of minimal cost vector for arc $\langle x^*,y^*\rangle\in A_G^*$ from x^* to y^*, $c^{*\min}(x^*,y^*)=\left\langle c_{z^*}^{*\min}(x^*,y^*)\right\rangle_{z^*\in\{v^*\in V^*:\langle v^*,x^*\rangle\in A^*\}}$, $c_{z^*}^{*\min}(x^*,y^*)$ – minimal of minimal cost from x^* to y^* when the predecessor of x^* is z^*,

$$c_{z^*}^{*\min}(x^*,y^*)=\min_{d(\cdot,\cdot)\in D^{\min}(W(x^*,z^*),W(x^*,y^*))}L(d(\cdot,\cdot))+\min_{d(\cdot,\cdot)\in D^{\min}(W(x^*,y^*),W(y^*,x^*))}L(d(\cdot,\cdot)) \qquad (3.2)$$

$c^{*\max}(x^*,y^*)$ – maximal of minimal cost vector for arc $\langle x^*,y^*\rangle\in A_G^*$ from x^* to y^*, $c^{*\max}(x^*,y^*)=\left\langle c_{z^*}^{*\max}(x^*,y^*)\right\rangle_{z^*\in\{v^*\in V^*:\langle v^*,x^*\rangle\in A^*\}}$, $c_{z^*}^{*\max}(x^*,y^*)$ – maximal of minimal cost for arc $\langle x^*,y^*\rangle\in A_G^*$ when the predecessor of x^* is z^*,

$$c_{z^*}^{*\max}(x^*,y^*)=\max_{d(\cdot,\cdot)\in D^{\min}(W(x^*,z^*),W(x^*,y^*))}L(d(\cdot,\cdot))+\max_{d(\cdot,\cdot)\in D^{\min}(W(x^*,y^*),W(y^*,x^*))}L(d(\cdot,\cdot)) \qquad (3.3)$$

$D^*(x^*,y^*)$ – set of paths between nodes x^* and y^* in graph G^*,

$d^*(x^*,y^*)=\left(x_0^*=x^*,x_1^*,x_2^*,...,x_{l^*(d^*(x^*,y^*))}^*=y^*\right)$ – element of $D^*(x^*,y^*)$ and $\underset{i=\overline{0,l^*(d^*(x^*,y^*))-1}}{\forall}\langle x_i^*,x_{i+1}^*\rangle\in A_G^*$,

$L^{*\min}(d^*(x^*,y^*))$ – cost of path $d^*(x^*,y^*)$ from x^* to y^*, which is based on $c^{*\min}(x^*,y^*)$,

$$L^{*\min}(d^*(x^*,y^*))=c_{p(x_0^*)}^{\min}(x_0^*,x_1^*)+\sum_{i=1}^{l^*(d^*(x^*,y^*))-1}c_{x_{i-1}^*}^{\min}(x_i^*,x_{i+1}^*) \qquad (3.4)$$

where $p(x_0^*)$ denotes the predecessor of x_0^* in G^* representing the "direction", from which we start path planning in x_0^* (we use this interpretation, for example when $\langle x^*,y^*\rangle\in A_G^*$ and x^*, y^* represent internal nodes of path $d^*(v^*,z^*)$; then $L^{*\min}(d^*(x^*,y^*))=c_{p(x^*)}^{\min}(x^*,y^*)$ and $p(x^*)$ denotes the predecessor x^* on path $d^*(v^*,z^*)$). If the information about $p(x_0^*)$ is unimportant then $p(x_0^*)=x_1^*$. Let us note that the interpretation of $p(x_0^*)$ allows us to write (3.4) as the sum of length of parts of path $d^*(x^*,y^*)$ as follows:

$$L^{*\min}(d^*(x^*,y^*)) = \sum_{i=0}^{l^*(d^*(x^*,y^*))-1} L^{*\min}(d^*(x_i^*,x_{i+1}^*)) = \sum_{i=0}^{l^*(d^*(x^*,y^*))-1} c_{p(x_i^*)}^{\min}(x_i^*,x_{i+1}^*)$$
$$p(x_i^*) = \begin{cases} x_{i-1}^*, & i>0 \\ x_1^*, & i=0 \end{cases} \tag{3.5}$$

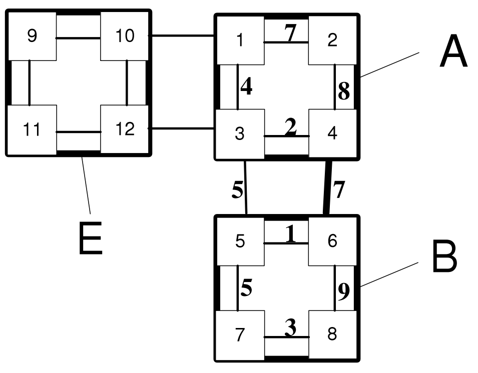
Without the presented interpretation of $p(x_0^*)$ the calculation of the length of $d^*(x^*,y^*)$ as the sum of the length of its parts like in (3.5) would be impossible. We can define $L^{*\max}(d^*(x^*,y^*))$ as the cost of path $d^*(x^*,y^*)$ from x^* to y^*, which is based on $c^{*\max}(x^*,y^*)$, analogically to (3.4):

$$L^{*\max}(d^*(x^*,y^*)) = c_{p(x_0^*)}^{\max}(x_0^*,x_1^*) + \sum_{i=1}^{l^*(d^*(x^*,y^*))-1} c_{x_{i-1}^*}^{\max}(x_i^*,x_{i+1}^*) \tag{3.6}$$

Finally, we denote with $d^{*\max}(x^*,y^*)$ the shortest path in G^* from x^* to y^* with $c^{*\max}(x^*,y^*)$ cost function and with $d^{*\min}(x^*,y^*)$ the shortest path in G^* from x^* to y^* with $c^{*\min}(x^*,y^*)$ cost function. For $d^{*\max}(x^*,y^*)$ and $d^{*\min}(x^*,y^*)$ following conditions are satisfied:

$$L^{*\max}(d^{*\max}(\cdot,\cdot)) = \min_{d^*(\cdot,\cdot)\in D^*(\cdot,\cdot)} L^{*\max}(d^*(\cdot,\cdot)), \qquad L^{*\min}(d^{*\min}(\cdot,\cdot)) = \min_{d^*(\cdot,\cdot)\in D^*(\cdot,\cdot)} L^{*\min}(d^*(\cdot,\cdot)),$$

where $D^*(\cdot,\cdot)$ describes the set of paths in G^* between pairs of b-nodes.



$$c^{*\min}(A,B) = \left\langle c_E^{*\min}(A,B), c_B^{*\min}(A,B)\right\rangle$$

$$c^{*\max}(A,B) = \left\langle c_E^{*\max}(A,B), c_B^{*\max}(A,B)\right\rangle$$

$W(A,E)$={1,3}; $W(A,B)$={3,4}; $W(B,A)$={5,6};

$$c_E^{*\min}(A,B) = \min_{d(\cdot,\cdot)\in D^{\min}(W(A,E),W(A,B))} L(d(\cdot,\cdot)) + \\ + \min_{d(\cdot,\cdot)\in D^{\min}(W(A,B),W(B,A))} L(d(\cdot,\cdot)) = 0+5=5$$

$$c_E^{*\max}(A,B) = \max_{d(\cdot,\cdot)\in D^{\min}(W(A,E),W(A,B))} L(d(\cdot,\cdot)) + \\ + \max_{d(\cdot,\cdot)\in D^{\min}(W(A,B),W(B,A))} L(d(\cdot,\cdot)) = 6+7=13$$

Fig. 3.2. The interpretation and calculation method of $c_E^{*\min}(A,B)$ and $c_E^{*\max}(A,B)$ as components of $c^{*\min}(A,B)$ and $c^{*\max}(A,B)$; calculation of $c_B^{*\min}(A,B)$ and $c_B^{*\max}(A,B)$ in accordance. As "border" nodes of A to B we have $W(A,B)$={3,4}

3.2.3. Decomposition Shortest Paths Algorithm (*DSP*)

3.2.3.1. The idea of the *DSP* algorithm

The branch-and-bound (decomposition) algorithm for shortest paths finding (*DSP* algorithm) consists of two main phases: (1) constructing graph G^* (steps 1-3); (2) finding the path from source *s* to destination *t* (steps 4-5). It uses Dijkstra's algorithm with *k*-ary heaps ($k = \max\{2, \lceil A / V \rceil\}$) (because graph *G* is sparse and *k*-ary heap is very effective (Tarjan, 1983)) and may be presented in 5 steps:

1. merge nodes from graph *G* (Fig. 3.1b) into *n* big nodes (b-nodes) as subgraphs of *G* (Fig. 3.1c) (*n* is the parameter of the algorithm);
2. set each of the subgraphs obtained in step 1 as b-nodes and set b-arcs in this graph as described by (3.1) obtaining graph G^* (Fig. 3.1d);
3. (a) for each $x^* \in V_G^*$ and for each $z^* \in V_G^*$ such that $\langle x^*, z^* \rangle \in A_G^*$ to determine the shortest path trees (*SPTs*) inside x^* for each $x \in W(x^*, z^*)$ as a source node;

   (b) calculate costs $c^{*\min}(\cdot,\cdot)$ and $c^{*\max}(\cdot,\cdot)$ for each arc of G^* using (3.2)-(3.3);
4. find the shortest path $d^{*\min}(x_s^*, x_t^*)$ and $d^{*\max}(x_s^*, x_t^*)$ in G^* with cost functions $c^{*\min}(\cdot,\cdot)$ and $c^{*\max}(\cdot,\cdot)$ (lower and upper restriction on length of the path from *s* to *t*) between such pairs x_s^*, x_t^* of b-nodes that $s \in x_s^*$, $t \in x_t^*$ (see Fig. 3.1d);
5. find shortest path from *s* to *t* (*s-t* path) inside subgraph generated by nodes of *G* belonging to b-nodes of $d^{*\min}(x_s^*, x_t^*)$ ($d^{*\max}(x_s^*, x_t^*)$):

   a) if $x_s^* = x_t^*$ then to find the shortest *s-t* path inside the subgraph of *G* generated by nodes belonging to $x_s^* = x_t^*$ (use paths calculated in step 3a);

   b) otherwise, if $x_s^* \neq x_t^*$, then *s-t* path may be found constructing the *DAG* with arcs directed from *s* to subset $W(x_0^* = x_s^*, x_1^*)$, then from $W(x_0^* = x_s^*, x_1^*)$ to $W(x_1^*, x_0^*)$, then from $W(x_1^*, x_0^*)$ to $W(x_1^*, x_2^*)$ etc. and lastly – from $W(x^*_{l^*(d^*(x_s^*, x_t^*))}, x^*_{l^*(d^*(x_s^*, x_t^*))-1})$ to *t* (Fig. 3.3). The arc cost in *DAG*, is between nodes *x* and *y*, and the length of the shortest path is calculated in step 3a.

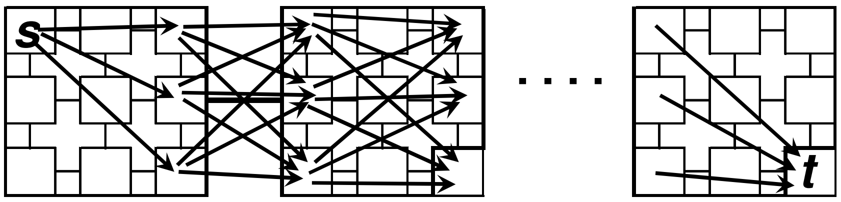


Fig. 3.3. Constructing *DAG* for the last step of the *DSP* algorithm. Firstly, arcs link *s* with nodes inside x_s^* bordering on x_1^*, then link previous nodes with nodes of x_1^* bordering on x_s^*, etc.)

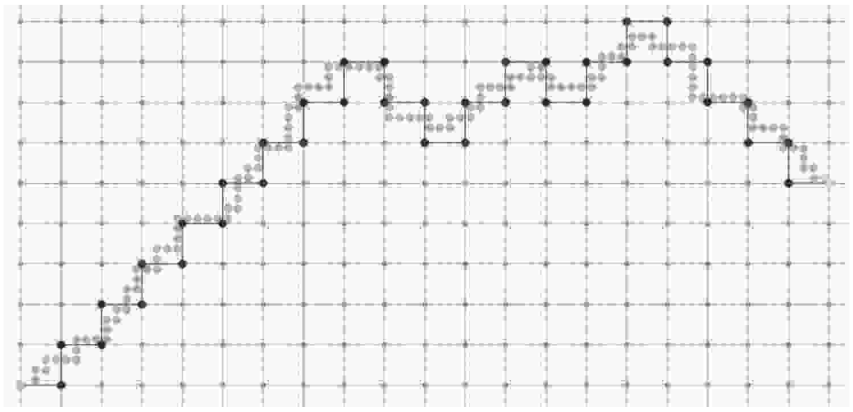


Fig. 3.4. An example of the path found in the *DSP* algorithm

An example of the path found by the *DSP* algorithm is presented in Fig. 3.4. Base mesh of nodes (from graph *G*) is drawn using the smallest square scale. B-nodes of graph G^* are drawn using the smallest gray circles (single b-node consist of 4x4 nodes from *G*), big gray circles denote path in *G* and big black circles – path in G^*.

3.2.3.2. Properties of the *DSP* algorithm

The *DSP* algorithm has some interesting properties. Theorem 3.1 shows lower and upper restriction on the length of the shortest path in *G* using the *DSP* algorithm. Theorem 3.2 shows the time and space complexity of the *DSP* algorithm.

Theorem 3.1

Let $L'(s, W(x_s^*, x_1^*))$ denote the length of the longest of the shortest paths from s to any node of $W(x_s^*, x_1^*)$ and x_1^* denote the direct successor of x_s^* on the path from x_s^* to x_t^*. For each s, t∈V_G and $x_s^*, x_t^* \in V_G^*$, $x_s^* \neq x_t^*$ such that $s \in x_s^*$, $t \in x_t^*$ the following formula is fulfilled:

$$L^{*\max}(d^{*\max}(x_s^*, x_t^*)) + L'(s, W(x_s^*, x_1^*)) \geq L(d^{\min}(s,t)) \geq L^{*\min}(d^{*\min}(x_s^*, x_t^*)) \quad (3.7)$$

Proof is presented in Appendix 3.A.1. Conclusions resulting from Theorem 3.1:

- if path from *s* to *t* exists in the *G* graph then path from x_s^* to x_t^* exists in the G^* graph and the *DSP* algorithm will find it;
- if *G*=G^* then the lower restriction equals the upper restriction (the *DSP* algorithm gives an optimal solution); otherwise, length $L(d^{\min}(s,t))$ of the shortest *s-t* path is restricted by the left and the right side of the inequality (3.7).

Theorem 3.2

Let digraph G=(V_G,A_G), s,t∈ V_G, $c : A_G \to R^+$ *and cardinal n representing the number of b-nodes in G* be given. Then the total time of the DSP algorithm (for preparing G* and finding shortest s-t path) is equal*

$$O\left(\sqrt{V^3/n}\log_k(V/n)+n\log_k n\right) \tag{3.8}$$

and the space $O\left(\sqrt{V^3/n}+A+V\right)$, *where* $k=\max\{2,\lceil A/V\rceil\}$.

Proof:

We must determine the complexity of each step of the algorithm.

Step 1. It can be done in $O(V)$ time;

Step 2. Each b-node has at most $\lceil V/n\rceil$ nodes and $N'=4\left(\left\lceil\sqrt{V/n}\right\rceil-1\right)$ "border" nodes. For each of the "border" nodes we must check at most 4 nodes of its neighbours to set the arc in G^*, we have to repeat it n times for each b-node, thus it requires time $O\left(n\sqrt{V/n}\right)$;

Step 3a. For a single b-node we have $N'=4\left(\left\lceil\sqrt{V/n}\right\rceil-1\right)$ border nodes and for each of them we have to determine the shortest paths tree using Dijkstra's algorithm with k-ary heap, $k=\max\{2,\lceil A/V\rceil\}\approx\max\{2,\lceil 4V/V\rceil\}=4$, thus for a single b-node it takes time $4\left(\left\lceil\sqrt{V/n}\right\rceil-1\right)\cdot O((V/n)\cdot\log_k(V/n))$. We calculate it n times and obtain a complexity of this step as follows: $O\left(\sqrt{V^3/n}\log_k(V/n)\right)$;

Step 3b. For a single b-node we have two cost vectors $c^{*\min}(\cdot,\cdot), c^{*\max}(\cdot,\cdot)$ each of them having at most 4 components. Calculation of each component takes time proportional to $2\sqrt{\lceil V/n\rceil}\sqrt{\lceil V/n\rceil}$, thus the total time is proportional to $8n\cdot 2\lceil V/n\rceil = O(V)$;

Step 4. G^* has n nodes and at most $4n$ arcs, thus calculation of the shortest path in G^* using Dijkstra's algorithm with k-ary heap takes time $O(n\log_k n)$, $k=\max\{2,\lceil 4n/n\rceil\}=4$;

Step 5. In the worst case $d^*(x_s^*,x_t^*)$ may have n b-nodes. By building *DAG* we can use only $2\sqrt{\lceil V/n\rceil}$ nodes and $\left(\sqrt{\lceil V/n\rceil}\right)^2=\lceil V/n\rceil$ arcs inside each b-node and $\left(\sqrt{\lceil V/n\rceil}\right)^2=\lceil V/n\rceil$ arcs between b-nodes (Fig. 3.3), thus number of arcs in the worst case is equal $2n\lceil V/n\rceil$. Using Bellman's algorithm (Bellman, 1958) for the shortest path in *DAGs* complexity is $O\left(n\sqrt{V/n}+V\right)$.

Let us note the 3rd step is a "bottleneck" of the algorithm (if $n \to 1$ then $\sqrt{V^3/n}\log_k(V/n) \to \sqrt{V^3}\log_k V$) and 4th step (if $n \to V$ then $n\log_k n \to V\log_k V$). Hence, the total time complexity of the *DSP* is $O\left(\sqrt{V^3/n}\log_k(V/n) + n\log_k n\right)$.

Space required is determined by step 3a; *SPT* for a single source node inside each b-node contains at most $\lceil V/n \rceil$ nodes (this is the number of nodes inside a single b-node), inside each b-node we determine *SPT* $4\left(\left\lceil \sqrt{V/n} \right\rceil - 1\right)$ times (for each border nodes as the root of *SPT*) and we have *n* b-nodes, hence it requires space proportional to $O(\sqrt{V^3/n})$. Moreover we need space for graph *G* ($O(A+V)$) and G^* ($O(4n+n)$), hence it requires $O(A+V)$ space. Thus the total space required by the *DSP* algorithm is $O(\sqrt{V^3/n} + A + V)$.

♦

3.2.4. Experimental Analysis of the *DSP* Algorithm

To examine the *DSP* algorithm we have used two models of the mesh *S* network with dimension 200x200 nodes and a structure similar to the one from Fig. 3.1b (only north-south-north, east-west-east arcs are permitted for each node, hence maximal number of arcs between two nodes is equal 4), (Tarapata & Godlewski, 2011c): S_1 – random arcs from network *S* have been cancelled and for each of the arc the random cost from the range $[1,4]$ has been set (after all the network has 119574 arcs); S_2 – all possible arcs between nodes have been conducted and for each of the arc random cost from the range $[1,4]$ has been set (after all, network have had 159200 arcs). Exactly 500 paths for randomly generated *s-t* pairs using the *DSP* and A* algorithms have been determined. Results for S_1 are presented in Table 3.1 and for S_2 – in Table 3.2.

Table 3.1. Experimental computation time and accuracy of the *DSP* algorithm for the S_1 network

dn	*Number of b-nodes (n)*	*Number of b-arcs (m)*	*G* generation time [s]*	*Time of DSP [s]*		*Time of A* [s]*		*Error [%]*	
				c^{min}	c^{max}	c^{min}	c^{max}	c^{min}	c^{max}
1	39 843	119 574	3.95	107.12	106.01	38.74	36.48	0	0
2	12 978	43 024	1.93	30.31	32.14	34.49	36.69	6.44	12.21
3	6 383	22 050	1.66	14.23	14.09	32.97	33.44	12.12	14.65
4	3 791	13 202	1.39	9.06	8.32	34.32	31.67	16.75	14.88
5	2 519	8 748	1.50	6.05	5.57	33.54	32.38	20.98	14.11
10	748	2 384	2.66	2.90	2.15	32.00	33.40	25.22	9.77
20	241	676	4.48	2.75	1.7	35.51	31.18	21.09	8.39
50	60	140	10.65	7.16	5.52	32.64	33.00	15.46	4.41
100	20	40	16.93	17.93	18.13	32.44	32.95	0.56	0.61

Table 3.2. Experimental computation time and accuracy of *DSP* algorithm for the S_2 network

dn	*Number of b-nodes (n)*	*Number of b-arcs (m)*	*G* generation time [s]*	*Time of DSP [s]*		*Time of A* [s]*		*Error [%]*	
				c^{min}	c^{max}	c^{min}	c^{max}	c^{min}	c^{max}
1	40 000	159 200	4.98	108.11	107.46	33.12	35.83	0	0
2	10 000	39 600	1.89	25.92	26.08	37.17	35.19	12.68	9.85
3	4 489	17 688	1.69	11.43	10.85	37.95	38.6	25.99	9.59
4	2 500	9 800	1.8	6.62	6.43	36.18	36.77	27.86	7.75
5	1 600	6 240	1.91	4.49	4.17	32.13	34.21	25.56	6.88
10	400	1 520	3.38	3.15	2.04	37.72	32.82	19.99	5.45
20	100	360	6.2	4.11	2.71	36.64	38.39	16.5	4.47
50	16	48	14.74	9.59	7.43	36.93	34.38	18.32	2.5
100	4	8	24.92	19.03	19.32	33.85	32.72	4.38	0.54

Columns in Table 3.1 and Table 3.2 contain (from the left): length of the b-node side (see Fig. 3.1c for an interpretation), number of b-nodes, number of b-arcs, generation time of G^* (total time for steps 1-3 of the *DSP* algorithm), total time of finding 500 paths by the *DSP* algorithm (total time for the steps 4-5 of the *DSP* algorithm, separately for c^{min} and c^{max}), total time of finding 500 paths with the A* algorithm (separately for c^{min} and c^{max}), *Error*=average absolute (in percent) difference between path lengths obtained from the *DSP* and optimal path lengths obtained from A* (separately for c^{min} and c^{max}). Results show that parameter *n* have a great impact on effectiveness and accuracy of the *DSP* algorithm. Both, extreme large and extreme small values of *n*, cause the deterioration of the *DSP* algorithm effectiveness and accuracy. Let us observe that for *n*=1 error of the algorithm is equal zero, but the computation time is significant greater than for A*. It results from the idea of the algorithm (G^* has a single b-node and only step 5th is realized). From the analysis results, that the *DSP* is more accurate for c^{max} than for c^{min}. This property is described in (Tarapata & Godlewski, 2011c; Godlewski, 2010).

Because the most complex steps of the algorithm (steps 1-3, "bottleneck") are done only one time (we build the b-graph only one time – initial pre-processing) then if we compute a single-pair of the shortest path many times it allows us to amortize time of the "bottleneck". In Fig. 3.5 we present graphs of calculation time (represented by the number of dominating operations) for finding *M* shortest paths using the *DSP* algorithm and the Dijkstra's algorithm between random pairs of nodes. It is easy to observe that the greater the value of *n* (with the same value of *V*) the smaller the number of shortest paths calculation to obtain a shorter time for the *DSP* algorithm than for the Dijkstra's algorithm. For example, to obtain the same calculation time for the *DSP* and the Dijkstra's algorithm for *V*=1024, *n*=4 we must find *M**=17 shortest paths (for *M*<17 the Dijkstra's algorithm is faster than *DSP*, otherwise the *DSP* algorithm is faster) and for *V*=1024, *n*=64 we must find only *M**=3 shortest paths (for *M*<3 the Dijkstra's algorithm is faster than the *DSP*, otherwise the *DSP* algorithm is faster). Taking this approach to the results given in

Table 3.1 for *dn*=5 we obtain the following data (for c^{max}): the time generation of G^* is equal TG^*=1.5 [s], computation times of finding 500 paths is equal 32.38 [s] for A* and 5.57 [s] for *DSP*; hence the average computation time for a single *s-t* shortest path calculation is equal: for *DSP* – *TD*=5.57/500=0.01114 [s], for A* – TA^*=32.38/500= 0.06476 [s]. We obtain that for M^*≥28 calculation time of *M** shortest paths using A* is greater than using *DSP*, because $M^* \cdot TA^* > TG^* + M^* \cdot TD$ that is $28 \cdot 0.06476 = 1.81328 > 1.5 + 28 \cdot 0.01114 = 1.81192$.

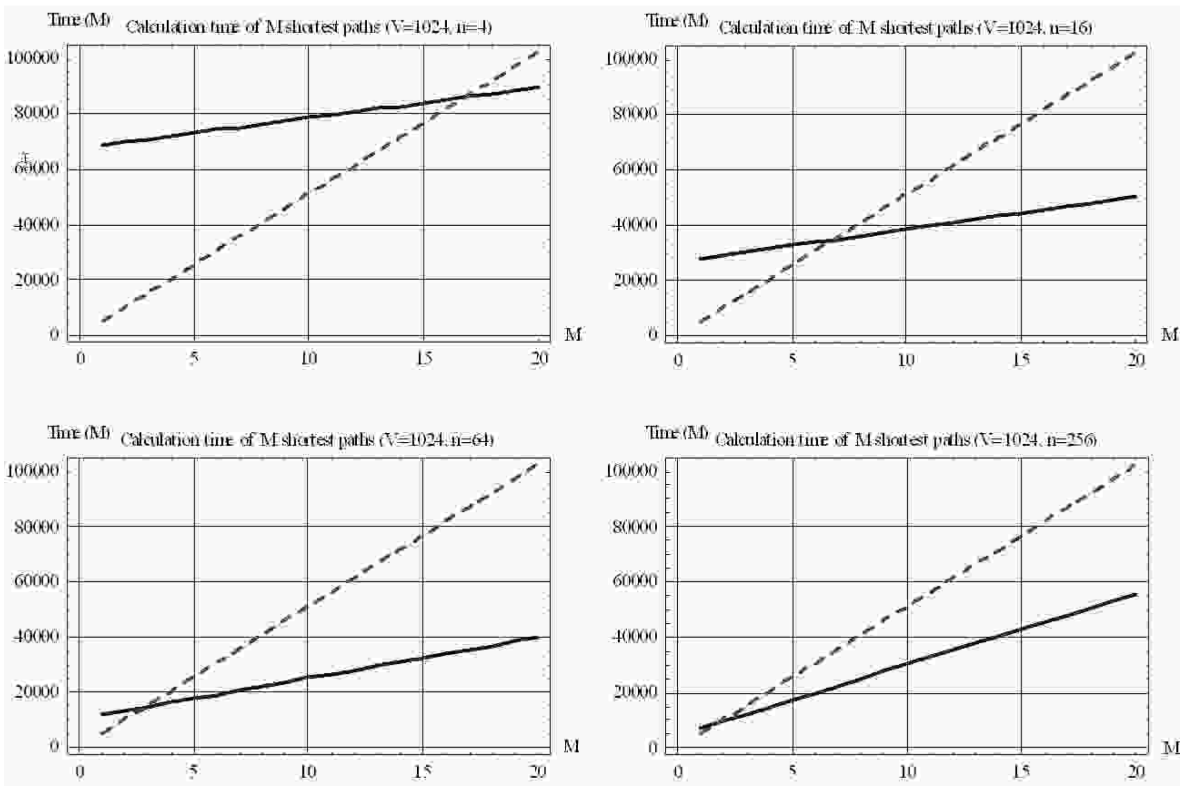


Fig. 3.5. Graphs of calculation time (represented by the number of dominating operations) for finding the M shortest paths using the *DSP* algorithm (continuous line) and the Dijkstra's algorithm (dashed line) between random pairs of nodes (*V*=1024, n=4, *n*=16, *n*=64, *n*=256)

Moreover, the comparison of the *DSP* with the *Hierarchical Algorithm* (*HA*) presented in (Chou *et al.*, 1998) has been conducted using the S_2 network (described at the beginning of this chapter). To understand the algorithm we present a short description of the *HA*. The *Hierarchical Algorithm* is designed to look for paths in large networks representing road networks. A road network in this model is divided into low level sub-networks (so-called micronetworks). If two nodes between which we find a path belonging to the same sub-networks then the path is looking for only in this sub-network (even then the optimal path uses nodes from other sub-networks). If two nodes belong to different sub-networks then the algorithm takes into consideration additional high level sub-network (so-called macronetwork, which is the sub-network of the original one). Each node from a macronetwork belongs to one or more micronetworks. Micronetworks may identify the network of local roads and macronetworks may identify highways and express roads. Looking for the shortest path between nodes belonging to different sub-networks relies on looking for a path from the initial micronetwork (source node belongs to this network) to any node *m* of the macronetwork, finding a path inside the macronetwork and then finding a path inside the destination micronetwork (destination node belongs to this network). If a macronetwork contains more than one node, then we must decide which node (*m*) must be choosen. We consider two strategies to this selection: *NearestHA* and *BestHA*. In the *NearestHA* strategy we choose the nearest macronetwork's node to the source/destination node in the micronetwork. In the *BestHA* strategy we choose such a node from macronetwork, for which length of the paths being found is the shortest. Path planning between micronetworks may be done using only the macronetwork.

In order to use *HA* we "cover" the S_2 network with macronetwork G^{**} (as mesh networks) with the length between macroarcs (arcs in the macronetwork) equals *dn*=5 (see Fig. 3.6). Exactly 500 paths for randomly generated *s-t* pairs using the *DSP* and the *HA* algorithms have been determined. Results presented in Table 3.3 show that the *HA* is faster than the *DSP*, but the error of the *HA* is significantly greater than for the *DSP*. These results show the high sensitivity of these algorithms to parameters. Inappropriate parameter settings (e.g. *n* for the *DSP* and *dn* for the *HA*) lowers the quality of solutions. The *DSP* algorithm is more tolerant to the initial model of a network. Moreover, in order to have correct computations the *HA* requires both a whole initial graph and all micronetworks and macronetworks to be strongly connected.

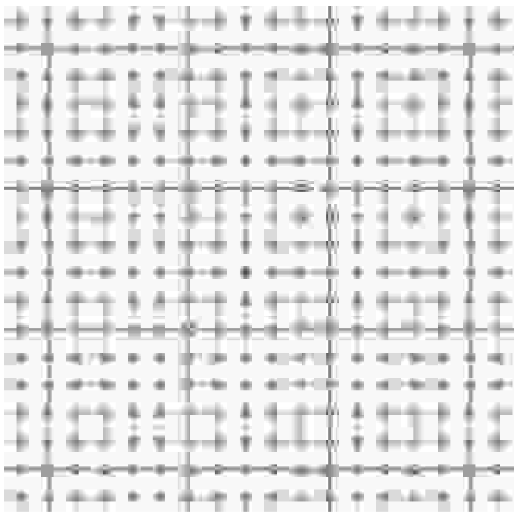


Fig. 3.6. Macronetwork G^{**} constructed for *dn*=5

Table 3.3. The comparison of *DSP* (a) and *Hierarchical Algorithm* (b)

a)

dn	*Number of b-nodes (n)*	*Number of b-arcs (m)*	*G* generation time [s]*	*Time of DSP [s]*		*Time of A* [s]*		*Error [%]*	
				c^{min}	c^{max}	c^{min}	c^{max}	c^{min}	c^{max}
4	2 500	9 800	1.8	6.62	6.43	36.18	36.77	27.86	7.75
5	1 600	6 240	1.91	4.49	4.17	32.13	34.21	25.56	6.88
10	400	1 520	3.38	3.15	2.04	37.72	32.82	19.99	5.45
20	100	360	6.2	4.11	2.71	36.64	38.39	16.5	4.47

b)

dn	*G** generation time [s]*	*Phase I time [s]*	*Time of HA [s]*		*Time of A* [s]*		*Error [%]*	
			Nearest HA	*Best HA*	*Nearest HA*	*Best HA*	*Nearest HA*	*Best HA*
4	168.30	30.40	0.08	0.21	39.17	40.90	25.42	21.44
5	105.70	15.63	0.09	0.19	39.32	41.54	28.65	22.68
10	28.16	12.67	0.09	0.15	40.19	40.46	38.83	22.89
20	7.80	39.40	0.08	0.63	38.58	39.04	44.60	19.30

Shown below are the advantages of using the *DSP* algorithm for finding the all-pairs shortest paths in network *G*. We can formulate acceleration functions $F_{Dijk}(V, n)$ and $F_{John}(V, n)$ as follows:

$$F_{Dijk}(V,n)=\frac{T_{Dijk}(V)}{T_{DSP}(V,n)} \qquad F_{John}(V,n)=\frac{T_{John}(V)}{T_{DSP}(V,n)} \tag{3.9}$$

where $T_{Dijk}(V)$, $T_{John}(V)$, $T_{DSP}(V,n)$ denotes, respectively, experimental average times of finding the all-pairs shortest paths in *G* with *V* nodes using: *V* times Dijkstra's algorithm with 4-ary heaps, Johnson's algorithm for sparse networks (Johnson, 1988), the *DSP* algorithm with *n* b-nodes.

Let the grid network with *V* squares (nodes) be given. We can formulate the following optimization problem: to find such a cardinal n^*, for which

$$F(V,n^*)=\max_{n\in\{1,\dots,V\}} F(V,n) \tag{3.10}$$

In Table 3.4 the experimental impact of *V* on n^* and $F(V,n^*)$ is shown. The value of n^* may be approximated by function $n^* \approx 1.87\cdot V^{0.34}$ and acceleration functions: $F_{Dijk}(V,n^*)\approx 0.39V^{0.67}$, $F_{John}(V,n^*)\approx 0.23V^{0.62}$, thus the average acceleration of *DSP* algorithm with relation to the Dijkstra's and Johnson's algorithm is $\cong O(V^{0.65})$.

Table 3.4. Experimental impact of the *V* on n^* and $F_{Dijk}(V, n^*)$, $F_{John}(V, n^*)$ for the all-pairs shortest paths problem for *V* various numbers of nodes from *G*

V	100	500	1 000	5 000	10 000	100 000	200 000	1 000 000
n^*	9	16	21	36	46	100	130	220
$F_{Dijk}(V,n^*)$	9	25	40	118	187	865	1 380	4 000
$F_{John}(V,n^*)$	5	12	18	49	75	320	495	1 400

3.2.5. Parallelization of the *DSP* Algorithm

The *DSP* algorithm can be very easily computed in parallel (Tarapata, 2010a). Because the *DSP* algorithm uses the Dijkstra's shortest path (or A*) algorithms (s.p.a.) as local-searching one, thus it is required to take into consideration the known results of the parallelization of this algorithm and the other s.p.a. There are many papers dealing with the problem of parallelization of s.p.a. Authors of the paper (Paige & Kruskal, 1985) propose a parallel version of the Dijkstra's algorithm, which uses a global reduction to extract the minimum distance node and then partitions the set of neighbours of that node among multiple processors. Using a binary heap-structured priority queue, this scheme has a running time of $O(A/p+k\cdot V)\cdot\log V$, where A and V are the number of edges (arcs) and nodes in the graph, p is the number of processors, and k is a constant representing the relative cost of communication vs. computation on the particular platform. A significant parallel speedup is possible only if $A/p \gg k$. Authors of the papers (Kumar *et al.*, 1994, sect. 7; Grama *et al.*, 2003, sect.10) show several approaches for parallelization of Dijkstra's s.p.a., in which execution time $T_{p,Dijk}$ of parallel the Dijkstra's algorithm using p processors is proportional to: $T_{p,Dijk}=(A/p)\cdot\log V+V\log p$ for the hypercube structure of the parallel computation system and $T_{p,Dijk}=(A/p)\cdot\log V+V\sqrt{p}$ for the mesh structure of the parallel computation system. Authors of the paper (Pantziou *et al.*, 1990) show efficient parallel algorithms, on the CREW PRAM[1] model, for generating a succinct encoding of all pairs shortest path information in a directed planar graph G with real-valued edge costs but no negative cycles. They assume that a planar embedding of G is given, together with a set of q faces that cover all the vertices. Then their algorithm runs in $O(\log^2 V+\log^3 q)$ time and employs $O(Vq)$ processors. Moreover, they present $O(\log^2 p)$ time, p-processor algorithms for various subproblems, including that of generating all pairs shortest path information in a directed outerplanar graph. Authors of other papers write about: parallelization of single-source s.p.a. (Atallah *et al.*, 1997; Crauser *et al.*, 1998; Foster, 1995, sect.3.9; Meyer & Sanders, 2001), parallelization of all-pairs s.p.a. (Atallah *et al.*, 1997; Foster, 1995, sect.3.9, Han *et al.*, 1997), parallelization of geometric and dynamic s.p.a (Lanthier *et al.*, 2003, Subramanian, 1995).

Analyzing steps of the *DSP* algorithm in chapter 3.2.3.2 it is easy to observe that the 3rd and the 4th steps are dominating from the point of view of algorithm complexity and they decide on the form of estimation (3.8): the 3rd step is dominating when $n<<V$ and the 4th step – when $n\to V$. Taking into consideration that best value n^* of n (from the point of view of time complexity) is proportional to

[1] Concurrent Read, Exclusive Write (CREW) Parallel Random Access Machine (PRAM).

$c \cdot \sqrt{V}$ with small nonnegative value of c (see chapter 3.2.3.2), for the big value of V we obtain that step 3rd is dominating. A very important problem from the point of view of parallelization effectiveness is to assign processors to the nodes (b-nodes) skilfully. Although we could assign each processor to subsets of nodes belonging to different b-nodes to try to increase effectiveness of the parallel *DSP* algorithm (*PDSP*), still this assignment may cause significant communication delays. The smaller migration of the processors between b-nodes the smaller the communications delay. The ideal solution from the point of view of minimizing communication delays is to minimize the number of assignments of processors to b-nodes. By doing this we minimize multiple copying subgraphs (b-nodes) to the local memory being used by processors. To explain these differences let us consider the structure of the G^* from Fig. 3.7a. For example, having p=2 processors it is better to assign the first processor to the left b-node, the second processor to the middle b-node (single copying to the local memory of the processor) and next (after calculating the shortest paths tree inside each b-node for each of the four nodes) to assign the first and the second processor to the different half of the right b-node. We then copy the subgraphs (b-nodes) for local memory of the processors only 4 times. In the worst case, if we omit the condition regarding minimizing migration of the processors between b-nodes, we may have a situation when each of the processors is assigned alternately for the left, middle and right b-node and we copy b-nodes for local memory of the processors V times (for each node inside each b-node). We consider two versions of parallelization: with and without parallelization of the Dijkstra's algorithm being used as a searching algorithm in the 3rd and 4th steps of the *DSP*. Let $t_{Dijk}(x) = x \cdot \log x$ describe the time complexity of the Dijkstra's algorithm in formula (3.8) and $N = \lceil V/n \rceil, N' = 4\left(\left\lceil \sqrt{V/n} \right\rceil - 1\right)$. Thus we can write (3.8) as follows: $T_1 = O\left(n \cdot N' \cdot t_{Dijk}(N) + t_{Dijk}(n)\right)$.

Theorem 3.3

The acceleration A(p) of the parallel DSP (PDSP) algorithm using p processors without parallelization of the Dijkstra's s.p.a. inside the DSP is as follows:

$$A(p) = \frac{T_1}{T_p} = \begin{cases} \dfrac{n \cdot N' \cdot t_{Dijk}(N) + t_{Dijk}(n)}{\lceil n/p \rceil \cdot N' \cdot t_{Dijk}(N) + t_{Dijk}(n)}, & \text{when } n \geq p \geq 1 \\[2ex] \dfrac{n \cdot N' \cdot t_{Dijk}(N) + t_{Dijk}(n)}{\lceil (n/p) \cdot N' \rceil \cdot t_{Dijk}(N) + t_{Dijk}(n)}, & \text{when } n \cdot N' > p > n \\[2ex] n \cdot N' \cdot t_{Dijk}(N) + t_{Dijk}(n), & \text{when } p \geq n \cdot N' \end{cases} \tag{3.11}$$

and no communication between processors is required.

Proof:

To prove the theorem we considered three cases of p values. We showed the T_p complexity of the *PDSP* algorithm with p processors determining the form of the T_p function. Let $T_B(p)$ describe the number of the Dijkstra's algorithm's parallel runs (d.a.p.r.) inside the 3rd step of the *DSP* algorithm using p processors. For p=1, $T_B(1)$ is equal $T_B(1) = n \cdot N'$.

If $n \geq p \geq 1$ then, in the first step, each of the p processors can be assigned to each of the p b-nodes of G^* (see Fig. 3.7). This step uses $\lfloor n/p \rfloor \cdot N'$ d.a.p.r. For the remaining $n - \lfloor n/p \rfloor \cdot p < n$ b-nodes we use $(n - \lfloor n/p \rfloor \cdot p) \cdot \lceil N'/p \rceil < N'$ d.a.p.r. Thus we can write that

$$T_B(p) = \lfloor n/p \rfloor \cdot N' + (n - \lfloor n/p \rfloor \cdot p) \cdot \lceil N'/p \rceil \leq (\lfloor n/p \rfloor + 1) \cdot N' \leq \lceil n/p \rceil \cdot N' \text{ d.a.p.r.}$$

Therefore the 3rd step of the *DSP* algorithm can be estimated using $\lfloor n/p \rfloor \cdot N' \leq T_B(p) \leq \lceil n/p \rceil \cdot N'$ d.a.p.r. and hence the estimation for both the 3rd and the 4th step of the *DSP* algorithm is as follows: $T_p = \lceil n/p \rceil \cdot N' \cdot t_{Dijk}(N) + t_{Dijk}(n)$. This estimation is the equality when (n mod p)=0 and otherwise ("$\leq$") it is an inequality. For example, using p=2 processors (see Fig. 3.7a) we first assign each of the p=2 processors to different b-nodes (dashed-line squares with 1 on the top) to calculate the shortest paths tree (*SPT*) for 4 nodes inside each b-node simultaneously using $\lfloor n/p \rfloor \cdot N'$=4 d.a.p.r. Next, for the remaining $n - \lfloor n/p \rfloor \cdot p = 1$ b-nodes we assign p=2 processors to the subsets of N'/p=2 nodes (dashed-line rectangles with 2 on the top) to calculate the *SPT* for 4 nodes inside the b-node simultaneously using $(n - \lfloor n/p \rfloor \cdot p) \cdot \lceil N'/p \rceil = 2$ d.a.p.r (total d.a.p.r. = 4+2). Using p=3 processors (see Fig. 3.7b) we assign each of the processors to each of the b-nodes (dashed-line squares with 1 on the top) to calculate the *SPT* for 4 nodes inside each b-node simultaneously using the total $\lfloor n/p \rfloor \cdot N' = 4$ d.a.p.r.

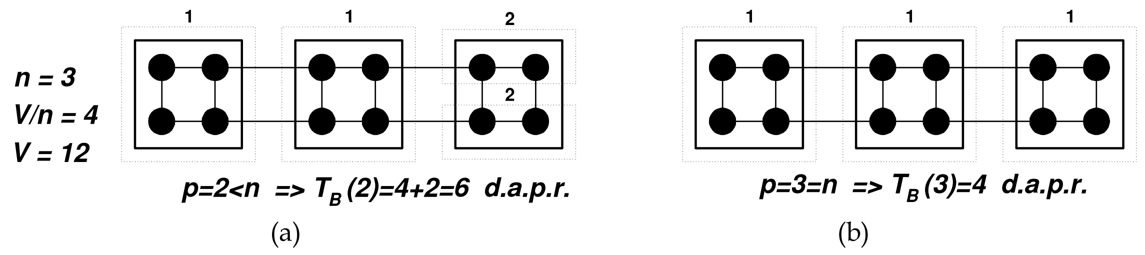


Fig. 3.7. Processors assignment for $n \geq p \geq 1$: (a) for p=2; (b) for p=3

If $n \cdot N' > p > n$ then we assign $\lfloor p/n \rfloor$ processors to each of the n b-nodes of the graph G^* and additionally 1 processor to each of the $p - n \cdot \lfloor p/n \rfloor$ b-nodes (see

Fig. 3.8). Thus, if (p mod n)≠0, then $p-n\cdot\lfloor p/n\rfloor$ b-nodes have $\lfloor p/n\rfloor+1$ assigned processors and $\lfloor p/n\rfloor$ processors otherwise, and they use $\lfloor (n/p)\cdot N'\rfloor$ d.a.p.r. Finally, for the remaining $n-(p-n\cdot\lfloor p/n\rfloor)$ b-nodes we assign processors using 1 d.a.p.r. Therefore the 3rd step of the *DSP* algorithm can be estimated using $T_B(p)=\lceil (n/p)\cdot N'\rceil$ d.a.p.r. and[2] $T_p=\lceil (n/p)\cdot N'\rceil\cdot t_{Dijk}(N)+t_{Dijk}(n)$. For example, using p=5 processors (see Fig. 3.8a) we first assign $n\cdot\lfloor p/n\rfloor=3\cdot 1$ processors to different b-nodes and additionally 1 processor for each of the $p-n\cdot\lfloor p/n\rfloor=2$ b-nodes (dashed-line squares with 1 on the top) to calculate *SPT* for 4 nodes inside each b-node simultaneously using $\lfloor (n/p)\cdot N'\rfloor=2$ d.a.p.r. Next, for the remaining $n-(p-n\cdot\lfloor p/n\rfloor)=3-2$ b-nodes we assign 2 processors: each for the nodes belonging to the remaining b-nodes (dashed-line rectangles with 2 on the top) to calculate *SPT* for two nodes inside the b-node simultaneously always using 1 d.a.p.r (total d.a.p.r. = 2+1). Using p=2n=6 processors (see Fig. 3.8b) we conduct analogical calculations when p=n=3 processors (see Fig. 3.7).

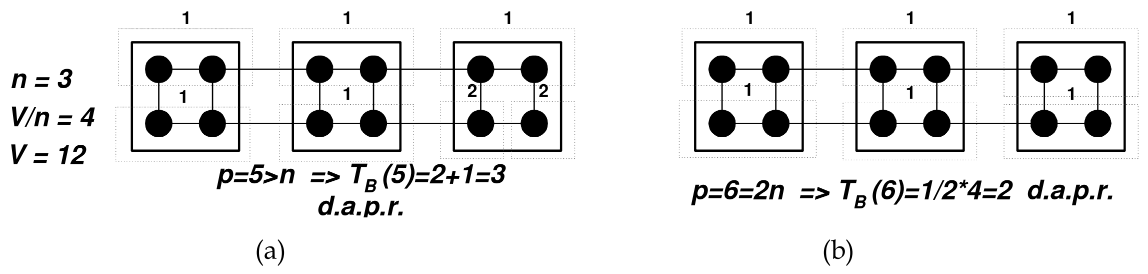

n = 3
V/n = 4
V = 12
1
1
1
1
1
2
2
p=5>n => T_B(5)=2+1=3 d.a.p.r.
(a)
1
1
1
1
1
1
p=6=2n => T_B(6)=1/2*4=2 d.a.p.r.
(b)

Fig. 3.8. Processors assignment for $nN'>p>n$: (a) for p=5; (b) for p=6

If $p>n\cdot N'$ then we assign $n\cdot N'$ processors to each of the $n\cdot N'$ nodes (N' processors to each of the n b-nodes) of the graph G^* using $T_B(p)=1$ parallel d.a.p.r. and $T_p=1\cdot t_{Dijk}(N)+t_{Dijk}(n)$.

♦

Because the acceleration function *A(p)* of the parallel algorithm using p processors is defined as (Foster, 1995; Kumar *et al.*, 1994; Grama *et al.*, 2003): $A(p)=T_1/T_p$ thus we obtain formula (3.11) using $T_1=n\cdot N'\cdot t_{Dijk}(N)+t_{Dijk}(n)$ and T_p defined as in the proof. Let us notice that, if $t_{Dijk}(n) \ll N'\cdot t_{Dijk}(N)$ then the acceleration function has the following form: $(n\ge p\ge 1)\Rightarrow A(p)=n/\lceil n/p\rceil$,

[2] Let us observe that $\lceil (n/p)\cdot N\rceil$ may not be equal $\lceil (n/p)\rceil\cdot N$.

$(n\cdot N'>p>n)\Rightarrow A(p)=n\cdot N'/\lceil (n/p)\cdot N'\rceil$, $(p\geq n\cdot N')\Rightarrow A(p)=n\cdot N'$. Effectiveness $E(p)$ of the *PDSP* is defined as (Foster, 1995; Grama *et al.*, 2003): $E(p)=A(p)/p$.

In order to consider the parallelization of the Dijkstra's algorithm inside the *DSP* algorithm we use two estimations for $T_{p,Dijk}$ time complexity of parallel Dijkstra's algorithm using the *p* processor given in (Kumar *et al.*, 1994, sect.7): $T_{p,Dijk}(V)=(1/p)\cdot A\cdot \log V+V\log p$ for the hypercube structure of the parallel computation system and $T_{p,Dijk}(V)=(1/p)\cdot A\cdot \log V+V\sqrt{p}$ for the mesh structure of the parallel computation system. Let the follwing be given:

$t^{N}_{Dijk,p}(N)=\left(\left\lfloor \dfrac{p}{n\cdot N'}\right\rfloor\right)^{-1}\cdot t_{Dijk}(N)+N\log\left\lfloor \dfrac{p}{n\cdot N}\right\rfloor$ and $t^{n}_{Dijk,p}(n)=(1/p)\cdot t_{Dijk}(n)+n\log p$

for the hypercube structure of the parallel computation system

and $t^{N}_{Dijk,p}(N)=\left(\left\lfloor \dfrac{p}{n\cdot N'}\right\rfloor\right)^{-1}\cdot t_{Dijk}(N)+N\sqrt{\left\lfloor \dfrac{p}{n\cdot N}\right\rfloor}$ and $t^{n}_{Dijk,p}(n)=(1/p)\cdot t_{Dijk}(n)+n\sqrt{p}$

for the mesh structure of the parallel computation system.

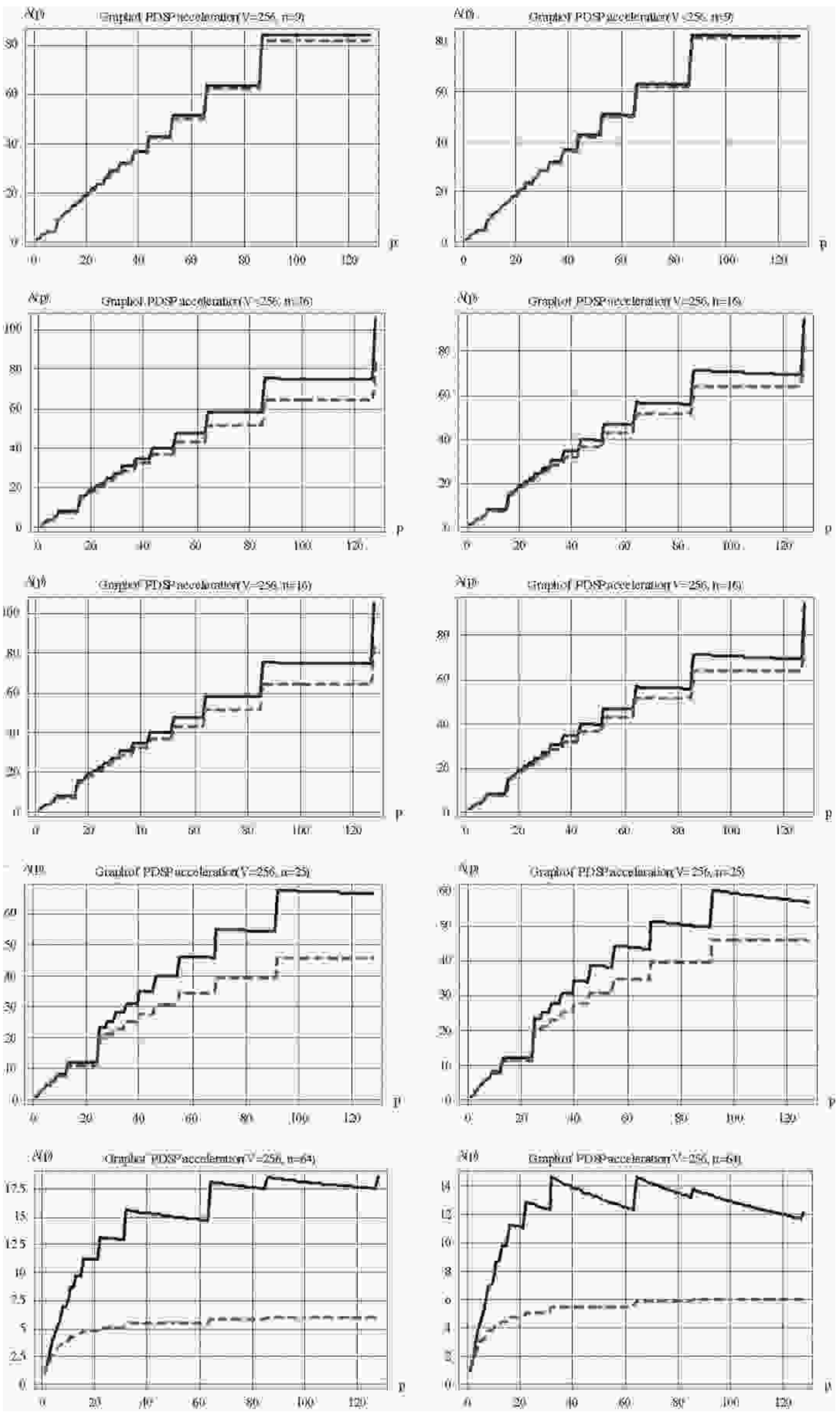
Theorem 3.4

The acceleration A(p) of the parallel DSP (PDSP) algorithm using p processors with the parallelization of the Dijkstra's s.p.a. is created by replacing in the denominators of (3.11) $t_{Dijk}(N)$ *with* $t^{N}_{Dijk,p}(N)$ *for* $p>nN'$ *and* $t_{Dijk}(n)$ *with* $t^{n}_{Dijk,p}(n)$ *for all p.*

Proof:

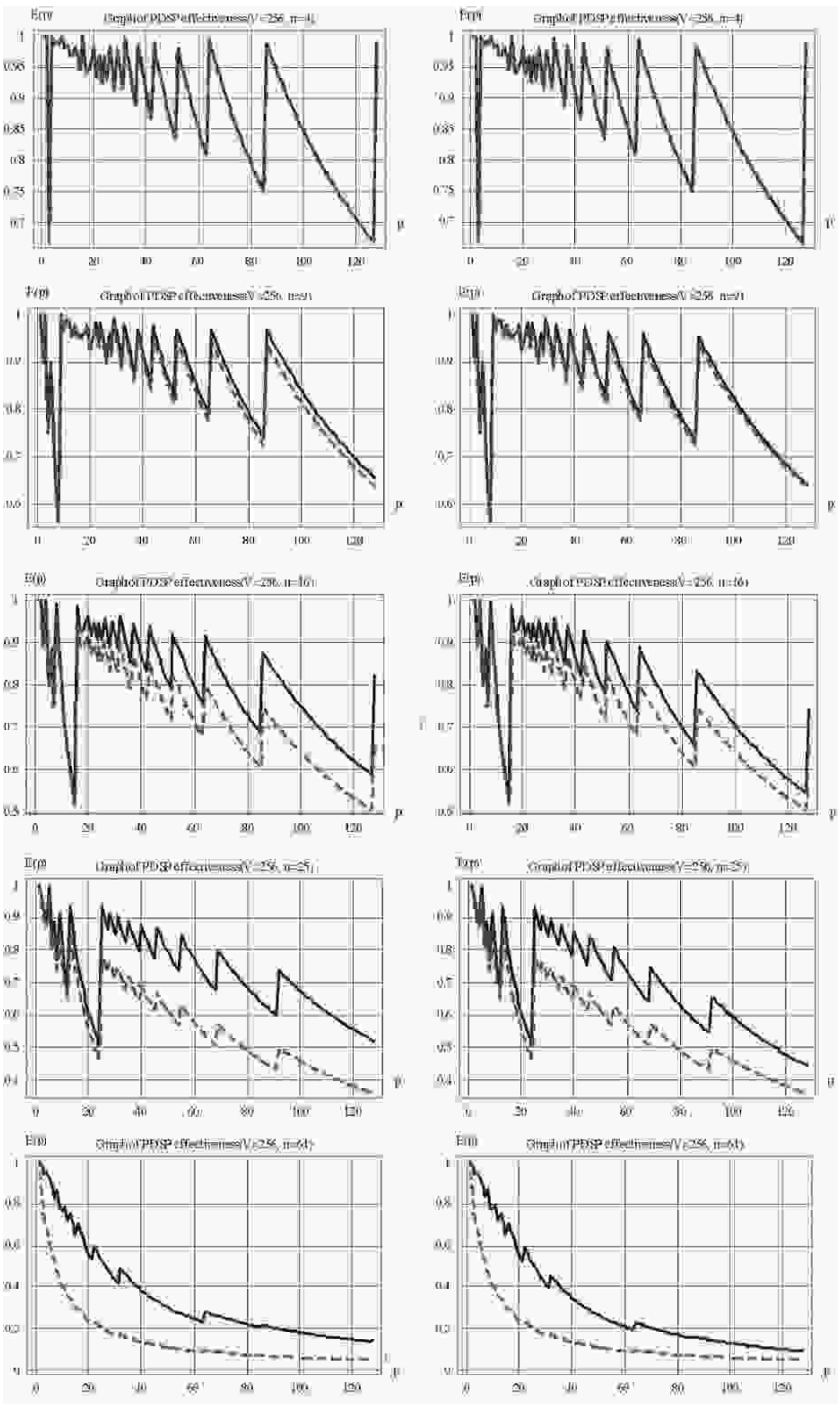
It has been shown that the $t_{Dijk}(n)$ estimation concerns the 4th step of the *DSP* algorithm, which is done after the 3rd step of the *DSP* so we can compute it in parallel independently of parallelization of the 3rd step. From the first element of the $T_{p,Dijk}$ formula results that having *p* processors we may calculate single shortest path *p* times faster (hence we have $(1/p)t_{Dijk}(n)$ in $t^{n}_{Dijk,p}(n)$) and for the second element of $T_{p,Dijk}$ – communications "costs" are proportional to $n\cdot\log p$. The form of the $t^{N}_{Dijk,p}(N)$ estimation results from the following reasoning: we can compute in parallel the Dijkstra's s.p.a. inside the 3rd step of the *DSP* only for $p>nN'$, because we use all processors when $p\leq n\cdot N'$ (see proof of the theorem 3.3). When the p mod $(nN')=0$ then we assign p/nN' processors for each of the *n* b-nodes, so each of the nodes inside each of b-nodes uses p/nN' processors to compute the *SPT* parallelly and compute it $\lfloor p/n\cdot N'\rfloor$ faster than having a single processor. Thus *p* from the $T_{p,Dijk}$ formula is equal to $\lfloor p/n\cdot N'\rfloor$ in the formula $t^{N}_{Dijk,p}(N)$.

♦



(a) (b)

Fig. 3.9. Graphs of simulation results of acceleration *A*(*p*) for the *PDSP* algorithm (*V*=256, $n\in\{4, 9, 16, 25, 64\}$) for the hypercube (a) and the mesh (b) structure of the parallel computation system. Continuous line concerns version of the *PDSP* with parallelization of the Dijkstra's s.p.a. and the dashed line – without parallelization of the Dijkstra's s.p.a.



(a) (b)

Fig. 3.10. Graphs of the simulation results of the *E*(*p*) effectiveness for the *PDSP* algorithm (*V*=256, *n*∈{4, 9, 16, 25, 64}) for the hypercube (a) and the mesh (b) structure of the parallel computation system. Continuous line concerns the version of the *PDSP* with parallelization of the Dijkstra's s.p.a. and dashed line – without the parallelization of the Dijkstra's s.p.a.

In Fig. 3.9 and Fig. 3.10 we present simulation results (done using MATHEMATICA 6.0 kernel) for acceleration (Fig. 3.9) and effectiveness (Fig. 3.10) of the *PDSP* algorithm for both cases defined in theorems 3.3 and 3.4 (when we parallelize and when we do not parallelize the Dijkstra's s.p.a. inside the *DSP* algorithm) and for two types of the structure of parallel computation systems: hypercube and mesh. We have conducted these researches for V=256 and different values of n: 4, 9, 16, 25 and 64. The greater n the better it shows the differences between effectiveness and acceleration (for the same p) for the case when we parallelize d.s.p.a. inside *DSP*. Moreover, it is visible that computations with parallelization of the Dijkstra's s.p.a. inside the *DSP* algorithm using the hypercube structure is a little more effective and we obtain a little better acceleration of the *PDSP* algorithm than using mesh structure.

3.2.6. Multiresolution Paths and the *DSP* Algorithm

Multiresolution environment is a nature of the hierarchical structure of military units and methods of their behaviours on a simulated battlefield. For a company level of units, greater precision of the terrain (environment) model is required than, for example, for the brigade level. In a battlefield simulation many models of the environment (terrain) representation is used (see chapter 2.1). The most popular are two representations: regular grid of terrain squares (Fig. 3.11a) and regular grid of terrain hexagons (Fig. 3.11b). The advantage of the first (square) terrain representation is especially visible in a multiresolution context (see Fig. 3.11c-e). The size of the terrain square may be dynamically changed and it depends on the required level of units. A square with a greater size than the basic size can be defined as a square matrix of basic-size squares (for example, in Fig. 3.11d each square has a size of 2x2 basic squares). Such a representation is not possible for hexagons, so square representation is more useful for multiresolution terrain modelling and path planning. In Fig. 3.11c-e an example is shown of path determination in the three-level graph: (c) the first level is the most detailed; (d) the second level is two times less detailed than the first; (e) the third level is four times less detailed than the first. These models may describe for example the platoon, company and battalion levels on the battlefield. Let us note that it is easy to obtain a multiresolution model of terrain by defining graph G^* recurrently. If we establish that graph G defines a terrain model of the first level (e.g. company level) than G^* defines a model of the second (or higher) level (e.g. battalion level). This reasoning may be used to increase or to decrease each required level of model resolution. Parameter n of the *DSP* algorithm ($n \in \{1,\ldots,V\}$) can be used to decide on the dimension of G^*. Then, the *DSP* algorithm may be used for finding multiresolution paths in such a multiresolution environment model. For example, in Fig. 3.11c $G^*=G$ and contains n=256 b-nodes (for the platoon level), in Fig. 3.11d G^* contains

n=64 b-nodes (for the company level) and in Fig. 3.11e G^* contains n=16 b-nodes (e.g. for the battalion level).

It is important to say that the presented method differs from very effective representations of terrain using quadtree (Kambhampati & Davis, 1986) because of two main reasons: (1) elements of the quadtree, which represent a terrain have a non-regular size, (2) in majority applications quadtree represents only a binary terrain with two types of regions: open (passable) and closed (impassable). This approach is very effective for mobile robots, but it is not adequate to represent multiresolution battlefield (Tarapata, 2003a).

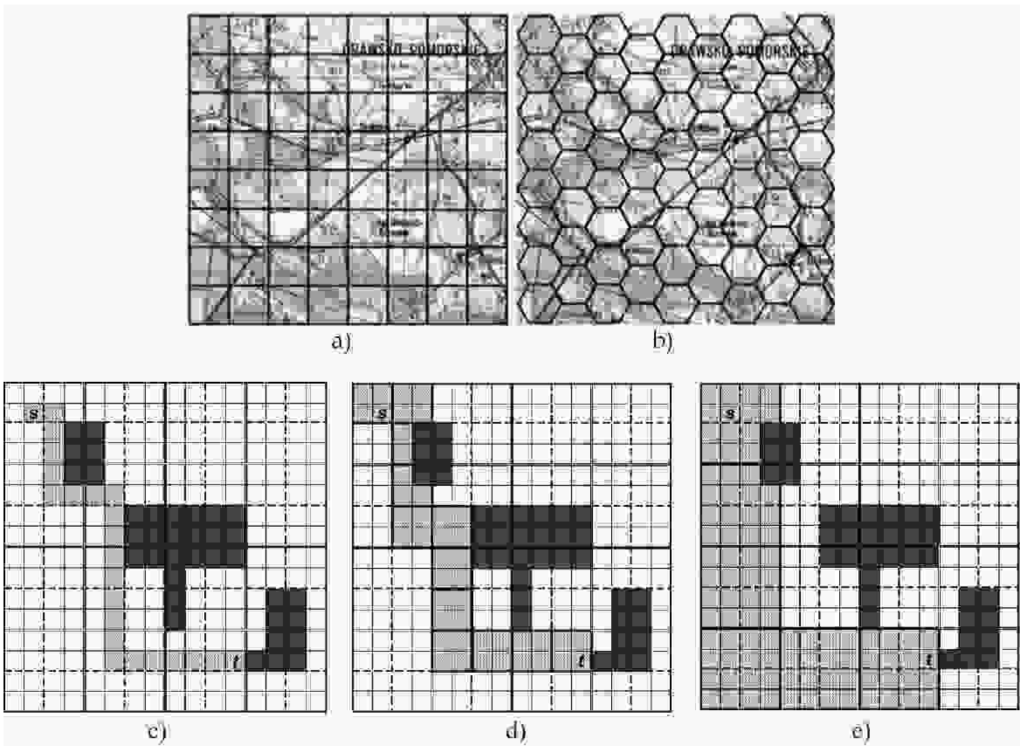


Fig. 3.11. Examples of terrain representation in a simulated battlefield: a) regular grid of terrain squares; b) regular grid of terrain hexagons; multiresolution shortest path from s to t using the *DSP* algorithm in G^*: c) $G^*=G$ contains 16×16 nodes; d) G^* contains 8×8 nodes; e) G^* contains 4×4 nodes

Let us note that the multiresolution approach for path planning represented by finding shortest paths in recurrently defined G^* can also be used for multistage path planning: first we can find a "rough" path $d^{*\min}(x_s^*, x_t^*)$ (or $d^{*\max}(x_s^*, x_t^*)$) – in a "rough" terrain represented by G^* (for example in Fig. 3.11e) and then we can find an accurate path in a more detailed environment (represented by G with small squares, Fig. 3.11c; more precisely: we find the shortest path from s to t (s-t path) inside the subgraph generated by nodes of G belonging to b-nodes of $d^{*\min}(x_s^*, x_t^*)$ (or ($d^{*\max}(x_s^*, x_t^*)$, see the 5th step of the *DSP* algorithm). This is an example of top-down modelling.

3.3. Multiobjective Paths Planning

3.3.1. Description of the Problem

The aim of this chapter is to analyze the complexity of the multiobjective (multicriteria) shortest paths (*MOSP*) problems and to show how we can use modifications and advantages of fast implementations of the Dijkstra's algorithm (using effective data structures such as the Fibonacci's heaps and d-ary heaps) in order to effectively and optimally solve the selected the *MOSP* problems.

The problem of finding the shortest path from a specified origin node to another node has been considered, traditionally, in the framework of the single objective optimization. More specifically, it is assumed that a value is associated to each arc (for example, the length or the travel time), and the goal is to determine the feasible path for which either the total distance or the total travel time is minimized (see chapter 3.1). In many real applications it is often found that a single objective function is not sufficient to characterize adequately the problem. In such a case the (*MOSP*) are used. There are many publications, which deal with these problems in two frequently used domains: computer networks (Cidon *et al.*, 1997; 1999; Grzech, 2002; Kerbache *et al.,* 2000; Silva & Craveirinha, 2004; Tarapata, 2005c) and transportation (Caramia & Guerriero; 2009; Dial, 1979; Halder & Majumber, 1981; Rana & Vickson, 1988; Fujimura, 1996; Modesti & Sciomachen, 1998). For instance, in transportation networks, a typical situation that can be adequately represented only considering more objectives is related to military route planning, where time, distance, ability to camouflage on the path must be taken into account at the same time (Tarapata, 2003a; 2007d). Another application, in which it is important to deal with several factors, is represented by path planning, where the goal is to find a navigation path for a mobile robot (Fujimura, 1996). In this case, the navigation path can be considered acceptable only if it satisfies multiple objectives, such as safety, time and energy consumption. In computer networks (as a special case of transportation networks) routing problems are one of the most essential applications of the *MOSP* problems. The most often used criteria of route selection depend on quality of service (*QoS*) (Silva & Craveirinha, 2004). These criteria are, for example, as follows: minimization of the number of lost packages; minimization of maximal delay time of packages; minimization of the number of disjoint routes or minimization of maximal transmission time for the disjoint routes (in case of disjoint routes); minimization of overload, measured with the mean value of traffic crossing by link; minimization of transmission time from source to destination; minimization of route length; minimization of probability of route unreliability or maximization of probability of route reliability. Single-criterion formulations of routing problems use previously defined criteria. The choice of an appropriate method for solving the defined problems depends on answers to the following questions: whether we want to determine routes statically (algorithms: Dijkstra's, Ford-Bellmann's, *PDM*, A*) or dynamically (adapting to current load) (Djidjev *et al.*, 1995); are there stochastic dependencies in the network (Sigal *et al.*, 1980; Korzan, 1982; 1983a; 1983b; Loui, 1983; Tarapata, 1999a; 2000e); whether we find the path for a single task or simultaneously for many tasks (e.g. through disjoint paths transmitting voice and picture or allocating channels in optical networks) (Li *et al.*, 1992; Schrijver & Seymour, 1992; Sherali *et al.*, 1998; Tarapata, 1999a); whether we plan to determine alternative paths (Golden & Skiscim, 1989). There are many papers which deal with the description of practical examples of using many criteria in routing problems (Kerbache & Smith, 2000; Silva & Craveirinha, 2004). For example, authors of the paper (Climaco *et al.*, 2002) consider a bicriterion approach for routing problems in multimedia networks. In practical considerations we often use contradicted criteria e.g. fast and reliable access to the services (risk-profit) (Korzan, 1982; 1983a; 1983b; Loui, 1983; Tarapata, 1999a; 2000e; 2007d). In such cases we can formulate and solve multicriteria optimization problem to support the decision of network designers (in computer or transportation networks) or administrators (traffic managers in transportation).

3.3.2. State of the Art in the Multiobjective Shortest Paths Problems (*MOSP*)

The *MOSP* problems are among the most tractable of NP-hard discrete optimization problems (Garey & Johnson, 1979). In the work of (Hansen, 1979) the existence was proved of a family of problems with an exponential number of optimal solutions. This implies that any algorithm solving the multiobjective shortest path problem is, at least, exponential in the worst case analysis. On the other hand some papers (Warburton, 1987; Vassilvitskii & Yannakakis, 2004; Tsaggouris & Zaroliagis, 2005) show that practical ε-approximate algorithms are generally limited either to problems having 2 or 3 criteria, or to problems requiring the ε-approximation of only certain restricted sets of efficient paths. One of the most popular methods of solving the *MOSP* problems is the construction of approximate Pareto curves (Papadimitriou & Yannakakis, 2000; Vassilvitskii & Yannakakis, 2004). Informally, a $(1+\varepsilon)$-Pareto curve P_ε is a subset of feasible solutions such that for any Pareto optimal solution, there exists a solution in P_ε that is no more than $(1+\varepsilon)$ away in all objectives. Papadimitriou and Yannakakis in their work (Papadimitriou & Yannakakis, 2000) show that for any multiobjective optimization problem there exists a $(1+\varepsilon)$-Pareto curve P_ε of (polynomial) size $\overline{\overline{P_\varepsilon}}$ $=O((4B/\varepsilon)^{N-1})$, where B is the number of bits required to represent the values in the objective functions (bounded by a polynomial in the size of the input), that can be constructed by $O((4B/\varepsilon)^d)$ calls to a "gap" routine that solves (in time polynomial in the size of the input and $1/\varepsilon$) the following problem: given a vector of values of *a*, either compute a solution that dominates *a*, or report that there is no solution better than *a* by at least a factor of $1+\varepsilon$ in all objectives (see definition 3.1 in chapter 3.3.3.1). Extensions to this method to produce a constant approximation to the smallest possible ($1+\varepsilon$)-Pareto curve for the cases of 2 and 3 objectives are presented in (Vassilvitskii & Yannakakis, 2004), while for N>3 objectives inapproximability results are shown for such a constant approximation. For the case of the *MOSP* (and some other problems with linear objectives), Papadimitriou and Yannakakis (Papadimitriou & Yannakakis, 2000) show how a "gap" routine can be constructed (based on a pseudopolynomial algorithm for computing exact paths), and consequently provide a *FPTAS* (*Fully Polynomial Time Approximation Scheme*) for this problem. Note that *FPTAS* for the *MOSP* problems were already known in the case of two objectives (Hansen, 1979), as well as in the case of multiple objectives in directed acyclic graphs (*DAGs*) (Warburton, 1987). In particular, the 2-objective case has been extensively studied (Ehrgott & Gandibleux, 2002), while for N>2 very little has been achieved; actually the results in (Warburton, 1987; Papadimitriou & Yannakakis, 2000; Tsaggouris & Zaroliagis, 2005) are the only and currently the best *FPTAS* known results. Let C^{max} denote the ratio of the maximum to the minimum edge weight (in any dimension), V denotes the number of nodes in a digraph, A denotes the number of arcs (edges) and N is the number of criteria. For the case of *DAGs* and N>2, the algorithm of (Warburton, 1987) runs in $O\left(VA\left(\frac{V(\log(VC^{\max}))}{\varepsilon}\right)^{N-1}(\log\frac{V}{\varepsilon})^{N-2}\right)$ time, while for N=2 this improves to $O\left(VA\frac{1}{\varepsilon}\log V\log(nC^{\max})\right)$. For N=2, a *FPTAS* can be created by repeated applications of a stronger variant of the "gap" routine – like a *FPTAS* for the restricted shortest path (*RSPP*) problem (Hassin, 1992; Lorenz & Raz, 2001; Ergun *et al.*, 2002). In (Vassilvitskii & Yannakakis, 2004) it is shown that this achieves a time of $O\left(VA\overline{\overline{P_\varepsilon^*}}(\log\log V+\frac{1}{\varepsilon})\right)$ for general digraphs and $O\left(VA\overline{\overline{P_\varepsilon^*}}/\varepsilon\right)$ for *DAGs*, where $\overline{\overline{P_\varepsilon^*}}$ is the size of the smallest possible ($1+\varepsilon$)-Pareto curve (which can be as large as $\log_{1+\varepsilon}VC^{\max}\approx\frac{1}{\varepsilon}\ln(VC^{\max})$). All these approaches deal typically with the single-pair version of the problem. Authors of the work (Tsaggouris & Zaroliagis, 2005) show a new and remarkably simple *FPTAS* for constructing a set of approximate Pareto curves for the single-source version of the *MOSP* problem in any digraph. For any N>1, their algorithm runs in time $O\left(VA\left(\frac{V\log(VC^{\max})}{\varepsilon}\right)^{N-1}\right)$ for general digraphs, and in $O\left(A\left(\frac{V\log(VC^{\max})}{\varepsilon}\right)^{N-1}\right)$ for *DAGs*. These results improve significantly upon previous approaches for general digraphs

(Golden & Skiscim, 1989; Hassin, 1992) and *DAGs* (Henig, 1985; Hassin, 1992), for all *N*>2. For *N*=2 their running times depend on ε^{-1}, while those based on repeated-*RSPP* applications (like in (Vassilvitskii & Yannakakis, 2004)) depend on ε^{-2}. Their approach for the *MOSP*, unlike previous methods that are based on converting pseudopolynomial time algorithms to *FPTAS* using rounding and scaling techniques, builds upon a natural iterative process that extends and merges sets of node labels representing partial solutions, while keeping them small by discarding some solutions in an error controllable way. One of the first papers, which dealt with the *MOSP* problems, was (Loui, 1983). The paper explores computationally tractable formulations of stochastic and multidimensional optimal path problems. A single formulation encompassing both problems is considered, in which a utility function defines preference among candidate paths. The result is the ability to state explicit conditions for exacting solutions using standard methods, and the applicability of well-understood approximation techniques. Korzan wrote three papers (Korzan, 1982; 1983a; 1983b), which deal with the shortest path problem in unreliable networks. In the first one he presents methods of determining the optimal path in unreliable directed networks under different assumptions concerning randomness of network elements. He assumes the vectoral objective function with two components: path length (e.g. time) and some measure of unreliability (e.g. probability of path "surviving"). An appropriate multioptimization problem and method for determining compromise path for this problem is described there. Some extensions of problems and their solving methods included there were discussed in further two papers (Korzan, 1983a; 1983b). In the papers (Tarapata, 1999a; 2000e) an optimization problem of a few tasks sending in a parallel or distributed computing system under conditions of unreliability of computers and lines is considered. As a model of the system a network is used with functions described on its nodes (time of task service in node and probability of nodes (computers) reliability) and arcs (time distances between nodes and probability of arc (line) reliability during transmission). The damaging process of a network element (node or arc) is begun: when a task starts its service in it (for a node) or its movement (for an arc) and it does not depend on the time, which elapsed from the start time of tasks sending (Tarapata, 1999a); when tasks start its service (or movement) in source nodes (Tarapata, 2000e). In the second case, the "time-life" distribution of network elements depends on the time, which elapsed from the start time of tasks sending. It may be explained by the fact that, for example, the probability of damaging an element of a computer network is grows in time. In the military communication systems the probability of destroying elements of the system depends on its working time (the longer the system working time the greater the possibility of the system locating and, in consequence, the higher the probability of annihilation of any elements of the system).

A problem for determining the best set of K>1 disjoint paths in an unreliable network is formulated as a two-criteria optimization problem, in which the first criterion is the time of sending the slowest task (or the sum of times of sending all tasks) being minimized and the second one – the probability of reliability of all (K>1) paths being maximized. An approximation algorithm to solve the optimization problem is shown. The algorithm generalizes the Dijkstra's shortest path algorithm in the case when we look for the K (K>1) disjoint paths in the network with two functions (probabilities and distances) described on the network nodes and arcs. Moreover, some conclusions concerning particular conditions, which the paths should satisfy, are given.

Table 3.5. Classification of the *Multiobjective Shortest Path Problem*s (*MOSP*)

Code of the problem	*References*
2-SUM/E/LC	(Tung & Chew, 1988; Brumbaugh-Smith & Shier, 1989; Skriver & Andersen, 2000)
2-SUM/E/LS	(Hansen, 1979)
2-SUM/E/2P,LC	(Mote *et al.*, 1991)
2-SUM/E/SP	(Martins & Climaco, 1981; Climaco & Martins, 1982; Huarng *et al.*, 1996)
2-SUM/E/DP	(Henig, 1985)
2-SUM/ Appr(E)/Appr	(Hansen, 1979)
1-SUM 1-max/E/SP	(Hansen, 1979; Pelegrin & Fernandez, 1998)
2-SUM/C/IA	(Current *et al.*, 1990)
2-SUM/U/SP	(Henig, 1985)
2-SUM/U/IA	(Murthy & Olson, 1994)
2-SUM/e/IA	(Coutinho-Rodrigues *et al.*, 1999)
2-SUM/C,SCH/LS	(Korzan, 1982; 1983b)
2-SUM/lex,SCH/LS	(Korzan, 1983a; 1983b)
3-SUM/E/LC	(Gabrel & Vanderpooten, 1996)
3-SUM/C/IA	(Gabrel & Vanderpooten, 1996)
Q-SUM/SE/SP	(Henig, 1985; White, 1987)
Q-SUM/E/LS	(Martins, 1984)
Q-SUM/E/LC	(Tung & Chew, 1992; Corley & Moon, 1985; Cox, 1984)
Q-SUM/E/DP	(Hartley, 1985; Kostreva & Wiecek, 1993)
Q-SUM/Appr(E),Appr(MO)/Appr	(Warburton, 1987)
Q-SUM/C/IA	(Henig, 1994)
Q-SUM/U/DP	(Carraway *et al.*, 1990)
Q-SUM/U/SP	(Modesti & Sciomachen, 1998)
Q-SUM/MO/DP,BB	(Rana & Vickson, 1988)
Q-SUM/MO/LC	(Murthy & Her, 1992)
Q-SUM/U,SCH/Appr	(Loui, 1983)
Q-SUM/MO,D,C,lex,SCH/Appr,LS	(Tarapata, 1999a; 2000e; 2005c; 2007d)

Generally, the multiobjective shortest path problem can be considered from the point of view of the following categories: number of criterions, type of problem (compromise solutions, lexicographic solutions, max-ordering problem, etc.), solution method (label setting or correcting algorithm, tabu search algorithm, simulated annealing algorithm and others). In Table 3.5 we classify the *MOSP* problems (as modification of classification proposed in (Ehrgott & Gandibleux, 2002)) using notation *X/Y/Z* where: *X* describes the number and type of objective functions (*X*=*Q* stands for an arbitrary number of objectives, e.g. 1-*SumQ*-max denotes a problem with the sum and *Q* bottleneck objectives), *Y* denotes the types of problems, *Z* denotes the types of solution methods. The entries for the *Y* position are as follows: *E* – finding the efficient set, *e* – finding a subset of the efficient set, *SE* – finding supported efficient solutions, *Appr*(*x*) – finding an approximation of *x*, *lex* – solving the lexicographic problem (preemptive priorities), *MO* – max-ordering problem, *U* – optimizing a utility function, *C/S* – finding a compromise respectively satisfying the solution, *D* – disjoint-path problem, *SCH* – stochastic problem. The entries for the *Z* position are as follows: *SP* – exact algorithm specifically designed for the problem, *LS/LC* – label setting or label correcting method, *DP* – algorithm based on dynamic programming, *BB* – algorithm based on branch and bound, *IA* – interactive method, *2P* – two phases method, *Appr* – approximation algorithm with worst case performance bound.

Other particular multiobjective path problems are presented in (Dial, 1979; Engberg *et al*, 1983; Halder & Majumber, 1981; Sancho, 1988; Wijeratne *et al.*, 1993).

3.3.3. Model of the *MOSP* Problem

3.3.3.1. Formulation of the *MOSP* problem

Let the directed graph $G=\langle V_G, A_G\rangle$ be given, where V_G – set of graph nodes, $V_G=\{1,2,...,V\}$, A_G – set of graph arcs, $A_G \subset \{\langle v,v'\rangle : v,v' \in V_G\}$, $\overline{\overline{A_G}}=A$. For example, in computer networks we have routers as nodes of *G* and physical links between routers as arcs of *G*. Generally, for each arc of *G* we may define arc functions $f_n(v,v')$, $n=1,\ldots,N$, which describe such characteristics of the arc $\langle v,v'\rangle \in A_G$ as: transmission time, distance, load, reliability, capacity, acceptable flows, etc. We assume that, there are *K* tasks, which we need to transport from source nodes i^s to destination ones i^d, $i^s=\left(i^s(1),i^s(2),...,i^s(k),...,i^s(K)\right)$, $i^d=\left(i^d(1),i^d(2),...,i^d(k),...,i^d(K)\right)$. For *K*=1 we have a classical case of routing for a single task. In some examples used in the chapter we use a computer network model such as *G* with predefined matrix $\mathbf{c}=[\mathbf{c}_{v,v'}]_{VxV}$, where $\mathbf{c}_{v,v'}=\left\langle c^1_{v,v'},c^2_{v,v'},...,c^k_{v,v'},...,c^K_{v,v'}\right\rangle$, $c^k_{v,v'}$ – nonnegative value describing transaction (transmission) time (cost) of the *k*-th task on the arc

$\langle v,v'\rangle \in A_G$ (when $v \neq v'$). Moreover, let $I_k(i^s(k),i^d(k))$ describe the simple path and $T_k(i^s(k),i^d(k))$ describe achieving times of nodes belonging to the path for the *k*-th task as follows:

$$I_k(i^s(k),i^d(k)) = \left(i^0(k) = i^s(k), i^1(k), \ldots, i^r(k), \ldots, i^{R_k}(k) = i^d(k)\right) \tag{3.12}$$

$$T_k(i^s(k),i^d(k)) = \left(\tau^0(k), \tau^1(k), \ldots, \tau^r(k), \ldots, \tau^{R_k}(k)\right) \tag{3.13}$$

where: $i^r(k)-$ the *r*-th node on the path for the *k*-th task; $\tau^r(k)-$ achieving time of the *r*-th node on the path for the *k*-th task,

$$\tau^r(k) = \sum_{m=1}^{r} c^k_{i^{m-1}(k),i^m(k)}, \quad r = \overline{1,R_k}, k = \overline{1,K} \tag{3.14}$$

We establish that if K=1 we omit index k (i.e. $i^r(1) \equiv i^r$, $\tau^r(1)) \equiv \tau^r$, etc.).

We describe by $M(i^s,i^d)$ the set of acceptable *K*-dimensional vectors of paths in *G* from i^s to i^d, and by $I(i^s,i^d)$ – the element of $M(i^s,i^d)$. It can be observed, that $I(i^s,i^d)$ is a vector which components are simple paths for each *k*-th task. We also establish, that $I \equiv I(i^s,i^d)$ (we omit i^s and i^d in the description). We assume that we have a *N*-component vector $F(I) = \langle F_1(I), F_2(I), \ldots, F_N(I)\rangle$ of criteria functions estimating vector of paths $I \in M(i^s,i^d)$. We have an arc function $f_n(v,v')$, $\langle v,v'\rangle \in A_G$, $n \in \{1,\ldots,N\}$, which will be used to calculate $F_n(I)$ (e.g. as a sum of values of $f_n(v,v')$ for arcs belonging to the path I). Thus, we can say that we have defined in the set $M(i^s,i^d)$ the vectoral objective function as follows:

$$F(I) = \langle F_1(I), F_2(I), \ldots, F_N(I)\rangle, \ I \in M(i^s,i^d) \tag{3.15}$$

The multicriteria shortest paths (*MOSP*) problem can be formulated as follows:

$$\langle M(i^s,i^d), F(I), R^D\rangle \tag{3.16}$$

where $R^D \subset Y^D(i^s,i^d) \times Y^D(i^s,i^d)$ is the domination relation in the criteria space

$$Y^D(i^s,i^d) = \left\{y(I) = F(I) = \langle F_1(I), F_2(I), \ldots, F_N(I)\rangle : I \in M(i^s,i^d)\right\} \text{ and} \tag{3.17}$$

$$R^D = \left\{\langle F(I_m), F(I_z)\rangle \in Y^D(\cdot,\cdot) \times Y^D(\cdot,\cdot) : \Psi(F(I_m), F(I_z)) = 1\right\} \tag{3.18}$$

$$\Psi(F(I_m), F(I_z)) = \begin{cases} 1 & \text{when } \mathrm{F}(I_m) \text{ "is better" than } \mathrm{F}(I_z) \\ 0 & \text{otherwise} \end{cases} \tag{3.19}$$

We can solve problem (3.16) using various methods of finding the so-called nondominated solutions. The set of nondominated results equals:

$$Y^{ND}(i^s,i^d) = \left\{ y(I) \in Y^D(\cdot,\cdot) \ : \ \sim \underset{\substack{z(I) \in Y^D(\cdot,\cdot) \\ z(I) \neq y(I)}}{\exists} \langle z(I), y(I)\rangle \in R^D \right\} \tag{3.20}$$

The set of nondominated solutions (paths) is determined as an inverse image of $Y^{ND}(i^s,i^d)$ as follows:

$$M^{ND}(i^s,i^d)=\{I\in M(i^s,i^d) : y(I)\in Y^{ND}(\cdot,\cdot)\}\tag{3.21}$$

In order to solve the *MOSP* problems other approaches are also used, e.g. the vector ε-domination (Warburton, 1987; Tsaggouris & Zaroliagis, 2005). The vector ε-domination method uses Definition 3.1.

Definition 3.1 (Warburton, 1987)

We say that vector $a=\langle a_1,a_2,...,a_N\rangle$ ε-dominates vector $b=\langle b_1,b_2,...,b_N\rangle$ for the fixed $\varepsilon\geq 0$ (we write: $a\overset{\varepsilon}{\leq}b$), if the following formula is satisfied:

$$\underset{n=\overline{1,N}}{\forall}\quad a_n\leq(1+\varepsilon)\cdot b_n\tag{3.22}$$

In some approaches it is additionally assumed that for at least one of the $n\in\{1,...,N\}$, e.g. n' we have $a_{n'}<(1+\varepsilon)\cdot b_{n'}$. It can be observed that for ε=0 this concept reduces to the usual notion of vector dominance. To use this approach we have to replace the domination relation (3.18) with the ε-domination relation $R^D_\varepsilon=\left\{\langle F(I_m),F(I_z)\rangle\in Y^D(\cdot,\cdot)\times Y^D(\cdot,\cdot):\ F(I_m)\overset{\varepsilon}{\leq}F(I_z)\right\}$ and we can solve a problem of finding the ε-shortest path which, according to (3.22), has cost no more than (1+ε) away from the optimal values for all objectives. Warburton in the paper (Warburton, 1987) studies methods for approximating the set of Pareto optimal paths in multiple-objective, shortest path problems. He gives the approximation methods that can estimate the Pareto optima to any required degree (ε) of accuracy. The basis of his results is that the proposed methods are "fully polynomial": they operate in time and space bounded by a polynomial in problem size and accuracy of approximation – the greater the accuracy, the more time required to reach a solution.

3.3.3.2. Example of the routing problem formulation as a two-criteria optimization problem

In the example of the routing problem formulation as the *MOSP* problem it is assumed that on each arc $\langle v,v'\rangle$ of the *G* graph we additionally define a function $q_{v,v'}(t)$ (identical for each task $k=\overline{1,K}$, so we omit *k* in the description of $q_{v,v'}(t)$), which describes the probability of the arc reliability at least at the time *t*: $q_{v,v'}(t)=\Pr\{\gamma_{v,v'}\geq t\}$, $\gamma_{v,v'}$ – nonnegative random variable representing "time-life" of the arc $\langle v,v'\rangle$. We assume that random variables $\gamma_{v,v'}$ are nonnegative and independent for each pair $\langle v,v' \rangle$ of arcs. Then for each vector of paths *I* in *G* we can define the probability that all *K* disjoint paths will "survive" as follows:

$$P\left(I(i^s,i^d)\right)=\prod_{k=1}^{K}\prod_{r=1}^{R_k} q_{i^{r-1}(k),i^r(k)}\left(c^k_{i^{r-1}(k),i^r(k)}\right) \tag{3.23}$$

Next we also define the time of achieving the destination nodes by all *K* tasks, as time of achieving the destination node by the most delayed task (3.24) or as a sum of achieving times of the destination nodes (3.25):

$$T\left(I(i^s,i^d)\right)=\max_{k\in\{1,\dots,K\}} \tau^{R_k}(k) \text{ or} \tag{3.24}$$

$$T\left(I(i^s,i^d)\right)=\sum_{k\in\{1,\dots,K\}} \tau^{R_k}(k) \tag{3.25}$$

Then the vectoral objective function (3.15) has the form of:

$$F(I)=\langle T(I),P(I)\rangle,\ I\in M(i^s,i^d), \tag{3.26}$$

i.e. $F_1(I)=T(I)$, $F_2(I)=P(I)$. Criteria space $Y^D(i^s,i^d)$ has the form:

$$Y^D(i^s,i^d)=\left\{F(I)=\langle T(I),P(I)\rangle : I\in M(i^s,i^d)\right\}, \tag{3.27}$$

and function (3.19) (which causes that relation (3.18) is a Pareto relation):

$$\Psi\left(F(I_m),F(I_z)\right)=\begin{cases}1 & \text{when } \left(T(I_m)<T(I_z)\wedge P(I_m)\ge P(I_z)\right)\vee \\ & \qquad \vee\left(T(I_m)\le T(I_z)\wedge P(I_m)>P(I_z)\right) \\ 0 & \text{otherwise}\end{cases} \tag{3.28}$$

We can equivalently define the problem formulated above as follows: to determine $I^*(i^s,i^d)\in M\left(i^s,i^d\right)$, for which

$$\begin{aligned}T^* &= T\left(I^*(i^s,i^d)\right)=\min_{I(i^s,i^d)\in M(i^s,i^d)} T\left(I(i^s,i^d)\right),\\ P^* &= P\left(I^*(i^s,i^d)\right)=\max_{I(i^s,i^d)\in M(i^s,i^d)} P\left(I(i^s,i^d)\right) \quad \text{or}\end{aligned} \tag{3.29}$$

$$\hat{P}^*=\min_{I(i^s,i^d)\in M(i^s,i^d)} \hat{P}\left(I(i^s,i^d)\right)=\min_{I(i^s,i^d)\in M(i^s,i^d)} 1-P\left(I(i^s,i^d)\right) \tag{3.30}$$

Generally, if the objective is to maximize one or more components of *F*(*I*) from (3.15), the *MOSP* algorithms can be applied to compute efficient paths only if *G* is acyclic (*DAG*). If *G* contains cycles and *N*=1 we solve the NP-hard longest path problem (for *N*>1 the problem is at least as difficult as for *N*=1) (Garey & Johnson, 1979). Therefore, we assume that all components of *F*(*I*) are minimized and all of these have nonnegative values.

3.3.4. Methods of Solving the *MOSP* Problems

3.3.4.1. Methods of solving single-criterion subproblems of the *MOSP* problem

Method of determining T^* and P^* from (3.29)-(3.30) depends on number K of tasks, for which we determine paths. If K=1, then we have the classical shortest paths problem in graph G for fixed pairs of nodes (i^s, i^d) with the arc function $c_{v,v'}$. This problem could be solved for the criterion function $T(I(i^s,i^d))$ using, e.g. algorithms described in chapter 3.1. When the arc function is nonadditive or nonlinear we can use the approach described by the authors in (Bernstein & Kelly, 1997; Cai *et al.*, 1997) or we can formulate a nonlinear optimization problem and solve it using Kuhn-Tucker's optimality conditions. For the function $\hat{P}(I(i^s,i^d))$ the approach presented in (Korzan, 1983b) could be used. Even though the function $\hat{P}(I(i^s,i^d))$ from (3.30) is multiplicative (multiplications of probabilities), then it is possible to obtain an additive form as follows:

$$\tilde{\hat{P}}\left(I(i^s,i^d)\right)=\sum_{k=1}^{K}\sum_{r=1}^{R_k}\left|\ln q_{i^{r-1}(k),i^r(k)}\left(c^k_{i^{r-1}(k),i^r(k)}\right)\right| \tag{3.31}$$

Defining the arc function as: $f_1(v,v')=\left|\ln q_{v,v'}\left(c_{v,v'}\right)\right|$ we can solve the problem (3.29)-(3.30) optimally using the Dijkstra's algorithm (because of function $f_1(v,v')$ is additive and nonnegative). The obtained solutions (i.e. $I^*(i^s,i^d)$) both for function $\hat{P}(I(i^s,i^d))$ and $\tilde{\hat{P}}(I(i^s,i^d))$ are identical. Other approaches to find the best path in stochastic graphs are considered in (Corea & Kulkarni, 1990; Cormican *et al.*, 1998; Sigal *et al.*, 1980; Korzan, 1982; 1983a; Loui, 1983; Tarapata, 1999a; 2000e).

The situation is more complicated when K>1. If we want to find disjoint routes for K tasks then even for K=2 and function $T(I(i^s,i^d))$ the problem is NP-hard (Schrijver & Seymour, 1992; Schrijver, 2004). Disjoint paths problems we will consider in chapter 3.4.

In further considerations in this chapter we assume that K=1. Let us note that for K=1 the objective functions (3.24) and (3.25) are equivalent. We also assume that $i_1^s=s_1$, $i_1^d=t_1$.

3.3.4.2. Method of compromise solutions

To find the compromise solution with parameter p≥1 we use the ε_p metric in the $Y^D(\cdot,\cdot)$ space:

$$\varepsilon_p(h^*,h(I))=\left\|h^*,h(I)\right\|_p=\sqrt[p]{\sum_{n=1}^{N}\left|h_n^*-h_n(I)\right|^p} \tag{3.32}$$

For the compromise result h^0 the following condition is satisfied:

$$\varepsilon_p\left(h^*,h^0(I)\right)=\min_{I\in M(i^s,i^d)}\varepsilon_p\left(h^*,h(I)\right) \tag{3.33}$$

The compromise solution $I^c\in M(i^s,i^d)$ is such that the formula (3.33) is fulfilled. Let us note that the metric (3.32) defines different distances from the "ideal" point (Ameljańczyk, 1984):

- for p=1 we obtain the sum of absolute deviations from ideal point (street metric);
- for p=2 we obtain the Euclidesian metric (in two-dimensional space = geometric distance between points) - "the best" compromise (Current *et al.*, 1990; Gabrel & Vanderpooten, 1996; Henig, 1994; Korzan, 1982; 1983b);
- for $p=\infty$ we obtain the Tchebycheff metric (minimization of maximal differences between "ideal" and actual value of criteria); this problem is also known as the max-ordering problem (Mote *et al.*, 1991; Rana & Vickson, 1988; Warburton, 1987).

To find compromise solution with parameter $p\geq1$ we use the metric ε_1 replacing $T(I)$ with $\overline{T}(I)$ and $P(I)$ with $\overline{P}(I)$. In order to find a compromise solution of the problem (3.16) with a vectoral objective function $F(I)=\langle T(I),P(I)\rangle$ we have to determine T^* and P^* using the method described in the previous chapter. Having T^* and P^* we can define $\overline{P}(I)=\frac{P(I)}{P^*}$, $\overline{T}(I)=\frac{T(I)}{T^*}$ by obtaining the normalized vector objective function:

$$h(I)=\left\langle\frac{T(I)}{T^*}\ ,\ \frac{P(I)}{P^*}\right\rangle \tag{3.34}$$

under the assumption, that $T^*\neq0$ and $P^*\neq0$. It can be observed that $\overline{T}(I)\geq1$ and $\overline{P}(I)\leq1$, $I\in M(\cdot,\cdot)$ so we can obtain the normalized ideal point $h^*=(1,1)$.

For example, for p=1 we obtain:

$$\varepsilon_1(h^*,h(I))=\left|1-\frac{T(I)}{T^*}\right|+\left|1-\frac{P(I)}{P^*}\right| \tag{3.35}$$

From the condition, that $1-\frac{T(I)}{T^*}\leq0$ and $1-\frac{P(I)}{P^*}\geq0$ results:

$$\varepsilon_1(h^*,h(I))=\frac{T(I)}{T^*}-1+1-\frac{P(I)}{P^*}=\frac{T(I)}{T^*}-\frac{P(I)}{P^*} \tag{3.36}$$

For the compromise h^0 result the following condition is satisfied:

$$\varepsilon_1(h^*,h^0(I))=\min_{I\in M(i^s,i^d)}\varepsilon_1(h^*,h(I))=\min_{I\in M(i^s,i^d)}\left[\frac{T(I)}{T^*}-\frac{P(I)}{P^*}\right] \tag{3.37}$$

For the compromise solution $I^c \in M(i^s, i^d)$ (with p=1) above formula is fulfilled.

However, since function $\frac{T(I)}{T^*} - \frac{P(I)}{P^*}$ has positive values then it is difficult to build an additive nonnegative arc function to calculate it. It is very inconvenient, because the Dijkstra's algorithm (as a classical algorithm solving shortest path problem) requires the values of the arc function to be nonnegative and additive (function $\varepsilon_1(h^*, h(I))$ is nonadditive because of multiplications during the calculation of the $\frac{P(I)}{P^*}$ value). The author of the paper (Korzan, 1982) shows that (for single task, i.e. K=1) if the arc function $q_{v,v'}(t)$ is in the form of $q_{v,v'}(t) = e^{-\lambda(v,v')\cdot t}$, $\lambda(v,v') > 0$, that is the probability function P from (3.23) equals:

$$P\left(I(i^s, i^d)\right) = \prod_{r=1}^{R_1} q_{i^{r-1}, i^r}\left(c^1_{i^{r-1}, i^r}\right) = \prod_{r=1}^{R_1} e^{-\lambda(i^{r-1}, i^r)\cdot c^1_{i^{r-1}, i^r}} = e^{\sum_{r=1}^{R_1} -\lambda(i^{r-1}, i^r)\cdot c^1_{i^{r-1}, i^r}} \tag{3.38}$$

then the maximization of $P\left(I(i^s, i^d)\right)$ is equivalent to minimization of $\beta\left(I(i^s, i^d)\right) = \sum_{r=1}^{R_1} \lambda(i^{r-1}, i^r)\cdot c^1_{i^{r-1}, i^r}$. In this case we can define a new normalized vectoral objective function $\hat{h}(I) = \left\langle T(I)/T^*\ ,\ \beta(I)/\beta^* \right\rangle$, where $\hat{h}(I) = \left\langle \overline{T}(I)\ ,\ \overline{\beta}(I) \right\rangle$, $\overline{T}(I) = T(I)/T^*$, $\overline{\beta}(I) = \beta(I)/\beta^*$ and ideal point $h^* = (1,1)$. Determining a new measure $\hat{\varepsilon}_1$ we obtain $\hat{\varepsilon}_1(h^*, \hat{h}(I)) = \left|1 - \overline{T}(I)\right| + \left|1 - \overline{\beta}(I)\right|$. But $1 - \overline{T}(I) \le 0$ and $1 - \overline{\beta}(I) \le 0$, so we obtain $\hat{\varepsilon}_1(h^*, \hat{h}(I)) = \overline{T}(I) - 1 + \overline{\beta}(I) - 1 = \overline{T}(I) + \overline{\beta}(I) - 2$. It can be observed that the function $\overline{T}(I) + \overline{\beta}(I) - 2$ has the minimum value for the same I as the function $\overline{T}(I) + \overline{\beta}(I)$, so the component (-2) may be omitted and we have:

$$\hat{\varepsilon}_1(h^*, \hat{h}^0(I)) = \min_{I \in M(i^s, i^d)} \hat{\varepsilon}_1(h^*, \hat{h}(I)) = \min_{I \in M(i^s, i^d)} \left[\overline{T}(I) + \overline{\beta}(I)\right] \tag{3.39}$$

The objective function from (3.39) is nonnegative and additive. Let us define the temporary function $H(I)$ as $H(I) = \overline{T}(I) + \overline{\beta}(I)$, so

$$\begin{aligned} H(I) &= \frac{T(I)}{T^*} + \frac{\beta(I)}{\beta^*} = \frac{1}{T^*}\sum_{r=1}^{R_1} c^1_{i^{r-1}, i^r} + \frac{1}{\beta^*}\sum_{r=1}^{R_1} \lambda(i^{r-1}, i^r)\cdot c^1_{i^{r-1}, i^r} = \\ &= \sum_{r=1}^{R_1}\left(\frac{1}{T^*} + \frac{1}{\beta^*}\cdot \lambda(i^{r-1}, i^r)\right)\cdot c^1_{i^{r-1}, i^r} \end{aligned} \tag{3.40}$$

In connection with the above we can define the problem of finding the compromise path $I^c \in M(i^s, i^d)$ with p=1 as follows: to determine $I^c \in M(i^s, i^d)$, such that

$$H(I^c) = \min_{I \in M(i^s, i^d)} H(I) \tag{3.41}$$

To solve the problem (3.41) optimally using the standard Dijkstra's algorithm we can use the following arc meta-function $mf(v,v')$:

$$mf(v,v') = \left(\frac{1}{T^*} + \frac{1}{\beta^*}\cdot\lambda(v,v')\right)\cdot c^1_{v,v'}, \quad \langle v,v'\rangle \in A_G \tag{3.42}$$

The definition presented above has one more interesting property: if for each arc $\langle v,v'\rangle \in A_G$ it is fulfilled that $\lambda(v,v') = \lambda > 0$ then $\beta\left(I(i^s,i^d)\right) = \lambda\cdot\sum_{r=1}^{R_1} c^1_{i^{r-1},i^r}$ and $\hat{h}(I) = \left\langle \frac{T(I)}{T^*}, \frac{\lambda T(I)}{\lambda T^*}\right\rangle$, so we can solve single-criterion problem with criterion *T*.

Generally, if arc functions $f_1, f_2, \ldots, f_N$ are nonnegative, additive (i.e. $F_n(I) = \sum_{r=0}^{R_1(I)-1} f_n(i_r(1), i_{r+1}(1))$) and all of these are minimized then the ε_1 measure from (3.32) (for p=1) has the form of:

$$\varepsilon_1(h^*, h(I)) = \sum_{n=1}^{N}\left|1 - \frac{F_n(I)}{F_n^*}\right| = \sum_{n=1}^{N}\left|1 - \frac{\sum_{r=0}^{R_1-1} f_n(v_r, v_{r+1})}{F_n^*}\right|, \tag{3.43}$$

where $F_n^* = \min_{I\in M(i^s,i^d)} F_n(I)$, $h(I) = \left\langle \frac{F_1(I)}{F_1^*}, \ldots, \frac{F_N(I)}{F_N^*}\right\rangle$, and $h^* = (\underbrace{1,1,\ldots,1}_{N \text{ times}})$. Because $1 - \frac{F_n(I)}{F_n^*} \le 0$ for all $n = \overline{1,N}$, so we can write that $\varepsilon_1(h^*, h(I)) = \sum_{n=1}^{N}\frac{F_n(I)}{F_n^*} - N$. It can be observed that function $\sum_{n=1}^{N}\frac{F_n(I)}{F_n^*} - N$ has the minimum value for the same *I* as function $\sum_{n=1}^{N}\frac{F_n(I)}{F_n^*}$, so the component (–*N*) may be omitted. In this case for the compromise result h^0 the following condition is satisfied (problem $CS_{p=1}$):

$$\varepsilon_1(h^*, h^0(I)) = \min_{I\in M(i^s,i^d)} \varepsilon_1(h^*, h(I)) = \min_{I\in M(i^s,i^d)} \sum_{n=1}^{N}\frac{F_n(I)}{F_n^*} \tag{3.44}$$

Thus, we can solve the problem $CS_{p=1}$ optimally using the standard Dijkstra's algorithm with the following arc metafunction $mf(v,v')$:

$$mf(v,v') = \sum_{n=1}^{N}\frac{f_n(v,v')}{F_n^*}, \quad \langle v,v'\rangle \in A_G \tag{3.45}$$

Proof of optimality of such an obtained solution is presented in the next chapter (with Theorem 3.5). For parameters p>1 it is impossible to obtain a nonnegative, additive, linear form of an arc function so it is rather impossible to solve the problem of finding a compromise solution optimally using Dijkstra's algorithm. In such cases the problem can be formulated as a quadratic programming problem (*p*=2) or max-ordering problem (*p*=∞) (Rana & Vickson, 1988; Warburton, 1987; Mote *et al.*, 1991). Method of compromise solutions with parameter $1 \le p < \infty$ guarantees obtaining nondominated solutions, i.e. $I^c \in M^{ND}(i^s, i^d)$ (Ehrgott, 1997; Martins & Santos, 1999).

In chapter 3.3.4.6 we define the $CS_{p=1}$ problem as a linear programming problem *MOSP_LP1* and *MOSP_LP2*, problem $CS_{p=2}$ as *MOSP_NP1* and problem $CS_{p=\infty}$ as *MOSP_NP2*.

3.3.4.3. Method with a metacriterion function

In this method we will construct a function, the so-called metacriterion function, which "merges" all criteria. There are two main approaches to define the metacriterion function: the first metacriterion function is in the form of a weighted average of criteria, in the second one we minimize maximal deviations of criteria values from its "ideal" values (an analogy to compromise the solution with parameter *p*=∞).

I. Metacriterion function in the form of weighted average of criterions with weights α_n, $n = \overline{1,N}$ is defined as follows (under assumptions that all criteria are minimized):

$$MF(I) = \sum_{n=1}^{N} \alpha_n \cdot F_n^*(I) \tag{3.46}$$

$$F_n^*(I) = \frac{F_n(I)}{F_n^*} = \frac{F_n(I)}{\min\limits_{I \in M(i^s,i^d)} F_n(I)} = \frac{\sum\limits_{r=0}^{R_1-1} f_n(v_r, v_{r+1})}{\min\limits_{I \in M(i^s,i^d)} F_n(I)}, \quad n = \overline{1,N} \tag{3.47}$$

where: $F_n^* > 0$, $f_n(\cdot,\cdot)$ describes the *n*-th arc function of *G*, $f_n : A_G \to R^+$, $n = \overline{1,N}$, R_1 describes the number of nodes belonging to path *I*. Frequently it is assumed that weights must satisfy following conditions: $\alpha_n \in (0,1)$, $n = \overline{1,N}$, $\sum_{n=1}^{N} \alpha_n = 1$. This guarantees obtaining nondominated solutions, i.e. $I^{MF} \in M^{ND}\left(i^s, i^d\right)$ (Ehrgott, 1997; Martins *et al.*, 1999).

The problem of finding an optimal solution (problem *MF_1*) can be formulated as follows: determine such a $I^{MF} \in M\left(i^s, i^d\right)$ that the following condition is fulfilled:

$$MF(I^{MF}) = \min_{I \in M\left(i^s,i^d\right)} MF(I) \tag{3.48}$$

We can solve this problem using the Dijkstra's algorithm with a single arc metafunction $mf(v, v')$ and with a metacriterion function *MF(I)*:

$$mf(v,v') = \sum_{n=1}^{N} \alpha_n \cdot \frac{f_n(v,v')}{F_n^*}, \quad \langle v,v' \rangle \in A_G \tag{3.49}$$

$$MF(I) = \sum_{r=0}^{R_1-1} mf(v_r, v_{r+1}) \tag{3.50}$$

Theorem 3.5

If arc functions f_1, f_2, ..., f_N, $f_i : A_G \to R^+$, $i = \overline{1,N}$ are additive then we can solve the problem (3.48) *optimally using the Dijkstra's algorithm with the arc meta-function* (3.49). *In this case the meta-function* (3.46) *is equal to meta-function* (3.50).

Proof:

When functions f_1, f_2, ..., f_N are nonnegative then the function (3.49) is nonnegative, and when functions f_1, f_2, ..., f_N are additive then the cost of path *I* is calculated as a sum of meta-costs of arcs belonging to path *I*. In this case assumptions of the Dijkstra's algorithm regarding the arc function (nonnegativity and additivity) are satisfied, so we can use this function as the arc function in the algorithm. Now, we will prove that *MF(I)=LF* from (3.46) is equal $\sum_{r=0}^{R_1-1} mf(v_r, v_{r+1}) = RG$ using (3.50). From (3.46) and (3.47) we obtain:

$$LF = MF(I) = \sum_{i=1}^{N} \alpha_i \cdot F_i^*(I) = \sum_{n=1}^{N} \alpha_n \cdot \frac{\sum_{r=0}^{R_1-1} f_n(v_r, v_{r+1})}{F_n^*} = \sum_{n=1}^{N} \sum_{r=0}^{R_1-1} \frac{\alpha_n}{F_n^*} \cdot f_n(v_r, v_{r+1}),$$

and from (3.49) and (3.50) we obtain

$$RG = \sum_{r=0}^{R_1(I)-1} mf(v_r, v_{r+1}) = \sum_{r=0}^{R_1-1} \sum_{n=1}^{N} \alpha_n \cdot \frac{f_n(v_r, v_{r+1})}{F_n^*} = \sum_{n=1}^{N} \sum_{r=0}^{R_1-1} \frac{\alpha_n}{F_n^*} \cdot f_n(v_r, v_{r+1}),$$

thus *LF=RG*.

♦

Let us note that arc function (3.45) is a special case of arc function (3.49) (all $\alpha_i = 1$), thus, problem (3.44) is a special case of problem (3.48).
The complexity of the algorithm is presented in Theorem 3.6.

Theorem 3.6

Complexity of the modified Dijkstra's algorithm (with Fibonacci's heaps) for solving problem (3.48) *using the arc metacriterion function* (3.49) *is equal* $O(N(V \log V + A) + NA)$.

Proof:

To calculate the arc metafunction (3.49) for each arc we must firstly solve the , shortest path problem N times for each criterion: it takes time proportional to $O\left(N(V\log V+A)\right)$ using Dijkstra's algorithm implemented with Fibonacci's heaps. Next, we have to separately calculate the metafunction (3.49) for each arc value; it takes a proportional time of $\Theta(NA)$ for all arcs. Using the Dijkstra's algorithm with the arc metafunction (3.49) we calculate the shortest path in a time of $O\left(V\log V+A\right)$, thus the total time of the algorithm for solving problem (3.48) is equal $O\left(N(V\log V+A)+NA\right)$.

♦

II. *Metacriterion function with minimization of maximal deviations of criteria values from their "ideal" values* can be defined using the following temporary function:

$$\overline{F}_n(I)=\frac{F_n^*}{F_n(I)}=\frac{\min\limits_{I\in M(i^s,i^d)}F_n(I)}{F_n(I)}=\frac{\min\limits_{I\in M(i^s,i^d)}F_n(I)}{\sum\limits_{r=0}^{R_1-1}f_n(v_r,v_{r+1})},\quad n=\overline{1,N} \tag{3.51}$$

Let us note that $\overline{F}_n(I)\in(0,1]$, $n=\overline{1,N}$, so the ideal point is equal 1. Now, we can define the *metacriterion function with minimization of maximal deviations of criteria values from their "ideal" values* (problem *MF_2*) as follows:

$$u\to\min \tag{3.52}$$

subject to

$$1-\overline{F}_n(I)\le u,\quad I\in M(i^s,i^d) \tag{3.53}$$

Additional variable u describes maximal deviation of values of criteria functions $\overline{F}_n(I)$ from their "ideal" values (i.e. 1). From the condition $\overline{F}_n(I)\in(0,1]$ results that $u\in[0,1)$. In chapter 3.3.4.6 we define this problem in details as a mathematical programming problem (*MOSP_NP*3).

We will show that the *MF_2* problem can be considered as a problem of finding (1+ε)-shortest path, $\varepsilon\ge 0$. Constraint $1-\overline{F}_n(I)\le u$ can be written as follows: $F_n(I)\le\frac{1}{1-u}\cdot F_n^*$. Taking into account the definition of the vector (1+ε)-dominance (see (3.22)) we obtain: $F_n(I)\le(1+\varepsilon)\cdot F_n^*$ that is $\frac{1}{1-u}=1+\varepsilon\Rightarrow\varepsilon=\frac{u}{1-u}$. Hence, $u\to\min$ is equivalent to $\varepsilon\to\min$, because ε is an increasing function of u. Therefore, the *MF_2* problem can be solved by finding (1+ε^*)-shortest path, where ε^* is the smallest value of ε such that (1+ε)-shortest path exists (we use the following property of the (1+ε)-shortest path: if any path I is the (1+ε)-shortest path then I is the (1+ε')-shortest path for each $\varepsilon' \geq \varepsilon$). If we set the precision for u to m decimal places (m is positive and an integer) then the *MF_2_half* algorithm is presented below.

Algorithm MF_2_half

```
L:=0; R:=$10^m$; u*:=infinity;
 WHILE |L-R|>1 DO
      u':= L + ceil((R-L)/2); u:=u'/$10^m$; $\varepsilon$:=u/(1-u);
      Determine (1+$\varepsilon$)-shortest path from $s_1$ to $t_1$;
      IF (1+$\varepsilon$)-shortest path from $s_1$ to $t_1$ exists THEN
           R:=u'; u*:=u;
      ELSE
           L:=u';
      END IF;
 END WHILE;
 RETURN u*;
```

If we denote with $T(\varepsilon)$ complexity of the algorithm of finding the (1+ε)-shortest path between s_1 and t_1 (see (Warburton, 1987; Papadimitriou & Yannakakis, 2000)), then the *MF_2_half* algorithm has a complexity $O\left(\log_2 10^m \cdot T(\varepsilon)\right)$ (because the idea is similar to binary-searching for value x in a sorted table with 10^m elements, where L and R denote, respectively, the left and right index of subtable range). For example, let the weighted graph be given in Fig. 3.12, s_1=1, t_1=5. The "ideal" vector of the criteria values is $c^* = \left(c_1^*, c_2^*, c_3^*\right) = (6,5,2)$. In the last column of Table 3.8 for each path I from s_1=1 to t_1=5 the smallest value of (1+ε) such that $F(I) \overset{\varepsilon}{\leq} c^*$ is calculated.

Let us set m=1 (we want to calculate u with a precision of one decimal place) for *MF_2_half* algorithm. In the first iteration, L=0, R=10, u'=5, u=0.5, ε=1. We see in Table 3.8 that path (e.g. pA) for which the (1+ε)≤2 exists, hence this path is the (1+(ε=1))-shortest path from s_1 to t_1 and R:=5, u^*:=0.5. In the second iteration, L=0, R=5, u'=2, u=0.2, ε=0.25. Because path (e.g. pA) for which (1+ε)≤1.25 exists, hence this path is the (1+(ε=0.25))-shortest path from s_1 to t_1 and R:=2, u^*:=0.2. In the third iteration, L=0, R=2, u'=1, u=0.1, ε=1/9. But the (1+(ε=1/9))-shortest path does not exist, hence L=1, R=2 and exit with u^*=0.2.

In chapter 3.3.4.6 we define problem *MF_1* as a linear programming problem (*MOSP_LP3*) and problem *MF_2* as *MOSP_NP3*.

3.3.4.4. Method with hierarchization of objective functions

In this approach we order criteria functions according to their importance (in the set of criteria function we set the lexicographic order), so F_1 describes the most important criterion, F_2 – the second criterion according to importance, etc. Solution $I^h \in M_{j\leq N}(i^s,i^d) \subset M(i^s,i^d)$ is found by solving the sequence of single-criteria optimization problems starting from the most important criterion (with index j=1, generating set $M_1(i^s,i^d)$), next taking into account the second criterion according to importance (generating set $M_2(i^s,i^d)$), etc. Calculations are continued as long as we achieve M_N or at the previous stage $s\leq N$ it occurs that $\overline{\overline{M_S}}=1$. Each M_j set narrows the previously obtained M_{j-1} set of acceptable solutions and it is recurrently defined:

$$M_j(i^s,i^d)=\begin{cases}\left\{I_j \in M_{j-1}(i^s,i^d) : F_j(I_j)=\min\limits_{I\in M_{j-1}(i^s,i^d)} F_j(I)\right\}, & \text{for } j=\overline{1,N}\\ M(i^s,i^d), & \text{for } j=0\end{cases} \tag{3.54}$$

The method of hierarchization of objective functions guarantees obtaining nondominated solutions, i.e. $I^h \in M^{ND}(i^s,i^d)$ (Ehrgott, 1997; Martins *et al.*, 1999).
For example, we considered the lexicographic solution (path) of the problem (3.16) with vectoral objective function $F(I)=\langle T(I),P(I)\rangle$, where P is defined as follows:

$$\begin{gathered}P(I(i^s,i^d))=\prod_{r=1}^{R_1} e^{-\lambda(i^{r-1},i^r)\cdot\sum_{k=1}^{r} c^1_{i^{k-1},i^k}}=\prod_{r=1}^{R_1} q_{i^{r-1},i^r}\left(\sum_{k=1}^{r} c^1_{i^{k-1},i^k}\right)\\ q_{i^{r-1},i^r}(t)=e^{-\lambda(i^{r-1},i^r)\cdot t}\end{gathered} \tag{3.55}$$

There is an interesting question: how to find a solution in the following order of criteria (3.29) importance: T, P? Korzan in (Korzan, 1983a) proved (for K=1) that if inside the set $M^{ND}(i^s,i^d)$ there exist many shortest paths, according to the criterion T with the same length T^* then all of these have the same value of the P criterion. Because of this fact any node x with the same value of T on the part of the path from s_1 can be considered at the next step of the Dijkstra's algorithm. Hence, we can use Dijkstra's algorithm with the modifications presented in Table 3.6, where: d(x) describes the value of function T for the path from s to x, c(x,y) is equivalent to $c_{x,y}$, p(x) describes the value of the P function for the path from s_1 to x, q(x,y,z) is equivalent to $q_{x,y}(z)$.
Modification of the Dijkstra's algorithm (*Dijkstra_Lex*2) has the same complexity as the original algorithm (with Fibonacci's heaps), that is $O(V\log V+A)$. Generally, finding lexicographic solutions (paths) is NP-hard (Garey & Johnson, 1979).

Table 3.6 Modification of the Dijkstra's algorithm for finding the lexicographic solution with *T, P* objectives

Standard Dijkstra's algorithm	*Dijkstra_Lex2 algorithm*
Dijkstra(	*Dijkstra_Lex2(*
$G=\langle V_G,A_G\rangle$, [c(u,v)]$_{VxV}$, s_1, t_1)	$G=\langle V_G,A_G\rangle$, [c(u,v)]$_{VxV}$,
	[q(u,v,z)]$_{VxVxT}$, s_1, t_1)
FOR EACH node v∈V_G **DO**	**FOR EACH** node v∈V_G **DO**
predecessor[v]:= null;	predecessor[v]:= null;
d[v]:= +infinity;	d[v]:= +infinity;
d[s_1]:= 0;	p[v]:= 0;
Q:= V_G;	p[s_1]:=1;
END FOR;	d[s_1]:= 0;
	Q:= V_G;
	END FOR;
WHILE Q ≠ null **DO**	**WHILE** Q ≠ null **DO**
u:= Extract_Min(Q);	u:= Extract_Min(Q);
/u is such a node that	*/u is such node that*
d[u]= min {d[v]:v∈Q}/	*d[u]= min{d[v]:v∈Q}/*
Q:= Q \ {u};	Q:= Q \ {u};
IF u=t_1 **THEN**	**IF** u=t_1 **THEN**
RETURN;	**RETURN;**
END IF;	**END IF;**
FOR EACH arc (u,v)∈A_G	**FOR EACH** arc (u,v)∈A_G
starting from u **DO**	starting from u **DO**
IF d[v]>d[u]+c(u,v) **THEN**	**IF** d[v]>d[u]+c(u,v) **OR**
	(d[v]=d[u]+c(u,v) **AND**
	p[v]<p[u]*q(u,v,d[v]))
	THEN
d[v]:= d[u] + c(u,v);	d[v]:= d[u] + c(u,v);
predecessor[v]:= u;	p[v]:= p[u] * q(u,v,d[v]);
	predecessor[v]:= u;
END IF;	**END IF;**
END FOR;	**END FOR;**
END WHILE;	**END WHILE;**

3.3.4.5. Method with threshold values of criteria (*Restricted Shortest Path Problem*)

Methods of threshold values (also known as restricted shortest paths problem (*RSPP*)) rely on the fact that some criteria functions have fixed critical values and they narrow the set of acceptable solutions. For example, problem (3.29) could be written as follows: to determine such a $I^*(i^s,i^d)\in M\left(i^s,i^d\right)$, that

$$P\left(I^*(i^s,i^d)\right)=\max_{I(i^s,i^d)\in M(i^s,i^d)} P\left(I(i^s,i^d)\right) \tag{3.56}$$

with an additional restriction: $T\left(I(i^s,i^d)\right) \le T_0$, where T_0 – fixed threshold value of criterion $T(\cdot)$. Warburton in (Warburton, 1987) showed an $O(V^2 Z \log V)$ algorithm for solving the *RSPP* problem for two objectives (with integer and positive values), where Z is the upper constraint on the value of the second objective (the first objective is minimized). In chapter 3.3.4.6 we define the *RSPP* problem as mathematical programming problem (*MOSP_LP4*).

3.3.4.6. Types of the *MOSP* problems defined as mathematical programming problems

For K=1 we will use the formulation of the *MOSP* problem as a linear programming problem as follows:

$$Cx \to \min \tag{3.57}$$

subject to

$$\begin{aligned} &Bx = d \\ &x \ge 0 \end{aligned} \tag{3.58}$$

Here $C = \left[c_{nj}\right]_{N\times A}$ is an objective matrix; $B = \left[b_{ij}\right]_{V\times A}$ is a transition matrix for graph G and: b_{ij}=1 when the j-th arc starts in the i-th node, b_{ij}= –1 when the j-th arc ends in the i-th node, b_{ij}=0 otherwise; $d = [d_i]_{V\times 1}$ is a column vector, which may have three values: d_i=1 when $i=i^s$, d_i=–1 when $i=i^d$; otherwise d_i=0; $x = \left[x_j\right]_{A\times 1}$, $x_j \in R^+ \cup \{0\}$; "min" describes minimum in the vectoral sense (in the sense of R^D relation). Each of the i-th node, $i = \overline{1,V}$ has its equivalent in the V_G set, each of the j-th arc, $j = \overline{1,A}$ has its equivalent in the A_G set and each c_{nj} cost for the j-th arc has its equivalent in the value of the arc function $f_n(v,v')$, $\langle v,v'\rangle \in A_G$. For the case of N=1, we have a classical definition of the shortest path problem as a linear programming problem (because of the total unimodularity of matrix B and vector d). Sometimes, we will use the extended, equivalent form of the problem (3.57)-(3.58):

$$\sum_{j=1}^{A} c_{nj} x_j \to \min, \quad n = \overline{1,N} \tag{3.59}$$

subject to

$$\begin{aligned} &\sum_{j=1}^{A} b_{ij} x_j = d_i, \quad i = \overline{1,V} \\ &x_j \ge 0, \quad j = \overline{1,A} \end{aligned} \tag{3.60}$$

The problem of finding a compromise solution with parameter p=1, however nonlinear in its nature, can be formulated as a linear programming problem. Using notations from (3.59)-(3.60) the metrics (3.32) can be written as follows:

$$\sum_{n=1}^{N}\left|1-\frac{1}{c_n^*}\sum_{j=1}^{A}c_{nj}x_j\right| \to \min \tag{3.61}$$

where $c_n^* \equiv F_n^*$. Let us accept following notations:

$$\bar{z}_n = \max\left\{0, 1-\frac{1}{c_n^*}\sum_{j=1}^{A}c_{nj}x_j\right\}, \quad n=\overline{1,N} \tag{3.62}$$

$$\bar{\bar{z}}_n = \max\left\{0, \frac{1}{c_n^*}\sum_{j=1}^{A}c_{nj}x_j - 1\right\}, \quad n=\overline{1,N} \tag{3.63}$$

Then for each $n=\overline{1,N}$ the following conditions are fulfilled:

$$\left|1-\frac{1}{c_n^*}\sum_{j=1}^{A}c_{nj}x_j\right| = \bar{z}_n + \bar{\bar{z}}_n \tag{3.64}$$

$$1-\frac{1}{c_n^*}\sum_{j=1}^{A}c_{nj}x_j = \bar{z}_n - \bar{\bar{z}}_n \tag{3.65}$$

$$\bar{z}_n \ge 0, \quad \bar{\bar{z}}_n \ge 0, \quad \bar{z}_n \cdot \bar{\bar{z}}_n = 0 \tag{3.66}$$

For this reason we obtain the following linear programming problem (*MOSP_LP1*):

$$\sum_{n=1}^{N}\bar{z}_n + \bar{\bar{z}}_n \;\to\; \min \tag{3.67}$$

subject to

$$1-\frac{1}{c_n^*}\sum_{j=1}^{A}c_{nj}x_j = \bar{z}_n - \bar{\bar{z}}_n, \quad n=\overline{1,N} \tag{3.68}$$

$$\bar{z}_n \ge 0, \quad \bar{\bar{z}}_n \ge 0, \quad \bar{z}_n \cdot \bar{\bar{z}}_n = 0, \quad n=\overline{1,N} \tag{3.69}$$

and (3.60)

We may omit conditions $\bar{z}_n \cdot \bar{\bar{z}}_n = 0, \quad n=\overline{1,N}$, but it can be shown that it does not extend the set of optimal solutions. The presented problem can be solved using the simplex algorithm. But the problem can be of large scale (number of variables equals N+A, number of boundaries equals N+V) and effectiveness of solving of this problem (using a simplex or ellipsoidal algorithm) is rather unacceptable.

According to the discussion conducted in chapter 3.3.4.2 and formula (3.44) the $CS_{p=1}$ problem can be also defined as follows (*MOSP_LP*2):

$$\sum_{n=1}^{N}\frac{1}{c_n^*}\sum_{j=1}^{A}c_{nj}x_j \to \min \tag{3.70}$$

subject to (3.60).

The $CS_{p=2}$ problem of finding a compromise solution with parameter p=2 (*MOSP_NP*1):

$$\sum_{n=1}^{N}\left(1-\frac{1}{c_n^*}\sum_{j=1}^{A}c_{nj}x_j\right)^2 \to \min \tag{3.71}$$

subject to (3.60). Unfortunately, the criterion function causes that the problem is nonlinear.

The $CS_{p=\infty}$ problem of finding a compromise solution with parameter p=∞ (*MOSP_NP*2), known as the max-ordering problem can be defined as follows:

$$\max_{n\in\{1,\ldots,N\}}\left|1-\frac{1}{c_n^*}\sum_{j=1}^{A}c_{nj}x_j\right| \to \min \tag{3.72}$$

subject to (3.60). The "max" in the criterion function causes that the problem is nonlinear. However, the problem can be formulated as linear ($u \to \min$, subject to: $\bar{z}_n + \bar{\bar{z}}_n \le u, \quad \forall n \in \{1,\ldots,N\}$, where $\bar{z}_n$, $\bar{\bar{z}}_n$ defined in (3.62) and (3.63)).

The method with the metacriterion function of type I (*MOSP_LP*3) is defined as follows:

$$\sum_{n=1}^{N}\frac{\alpha_n}{c_n^*}\sum_{j=1}^{A}c_{nj}x_j \to \min \tag{3.73}$$

subject to: $\sum_{n=1}^{N}\alpha_n = 1$, $\alpha_n \ge 0, \quad n=\overline{1,N}$ and (3.60).

To define the *MOSP* problem with the metacriterion function of type II, let us note that function $\bar{F}_n(I)$ from (3.51) is equivalent to $\frac{c_n^*}{\sum_{j=1}^{A}c_{nj}x_j}$, hence we obtain (*MOSP_NP*3):

$$u \to \min \tag{3.74}$$

subject to

$$1-\frac{c_n^*}{\sum_{j=1}^{A}c_{nj}x_j} \le u, \quad n=\overline{1,N} \tag{3.75}$$

and (3.60).

The first type of constraint causes that the problem is nonlinear.

The method with critical values of criteria (*MOSP_LP*4) known also as the restricted shortest path problem can be formulated as follows:

$$\sum_{j=1}^{A} c_{Lj} x_j \to \min \tag{3.76}$$

subject to

$$\sum_{j=1}^{A} c_{ij} x_j \le g_i, \quad i = \overline{1,N},\ i \ne L \tag{3.77}$$

and (3.60)

where $g = (g_1, g_i, \dots, g_N)_{i \ne L}$ describes the threshold values of each of the criteria, and L denotes the index of the criterion to minimize. Let us note that if any component of g is not an integer then the constraint $x_j \ge 0,\ \ j = \overline{1,A}$ must be replaced by $x_j \in \{0,1\},\ \ j = \overline{1,A}$.

In Table 3.7 we present properties of the *MOSP* problems formulated as mathematical programming problems.

Table 3.7. Properties of the *MOSP* problems formulated as mathematical programming problems

Problem	*Type of mathematical programming problem*	*Number of decision variables*	*Number of constraints*
MOSP_LP1	*Linear*	2*N*+*A*	*V*+*N*
MOSP_LP2	*Linear*	*A*	*V*
MOSP_LP3	*Linear*	*A*	*V*
MOSP_LP4	*Linear*	*A*	*V*+*N*-1
MOSP_NP1	*Nonlinear*	*A*	*V*
MOSP_NP2	*Nonlinear*	*A*	*V*
MOSP_NP3	*Nonlinear*	*A*+1	*V*+*N*

3.3.4.7. Example of the GAMS model for the *MOSP_LP*3 problem

The source code of the the GAMS[3] model for solving the *MOSP_LP*3 problem (the first row in Table 3.9, equation (3.73) and (3.60)) for the *G* graph from Fig. 3.12 is presented below. We set the following equivalence between notations being used in the *MOSP_LP*3 model and in the source code of the GAMS model (notation *x*≡*y* describes that *x* in the GAMS model is equivalent to *y* in the *MOSP_LP*3 model): c(n,j) $\equiv c_{nj}$, b(i,j) $\equiv b_{ij}$, c_opt(n) $\equiv c_n^*$, alfa(n) $\equiv \alpha_n$, x(j) $\equiv x_j$, d(i) $\equiv d_i$.

[3] General Algebraic Modelling System (Rosenthal, 2010).

```
Sets
i                 nodes of the graph G
/1, 2, 3, 4, 5/

j                 arcs of the graph G
/1, 2, 3, 4, 5, 6, 7, 8, 9, 10/

n             criteria
/1, 2, 3/

Parameters
c(n,j)            cost matrix of the graph G;
c('1','1')= 1;
c('1','2')= 3;
c('1','3')= 5;
c('1','4')= 3;
c('1','5')= 2;
c('1','6')= 4;
c('1','7')= 2;
c('1','8')= 6;
c('1','9')= 3;
c('1','10')= 2;
c('2','1')= 3;
c('2','2')= 4;
c('2','3')= 2;
c('2','4')= 2;
c('2','5')= 4;
c('2','6')= 2;
c('2','7')= 3;
c('2','8')= 2;
c('2','9')= 2;
c('2','10')= 5;
c('3','1')= 1;
c('3','2')= 1;
c('3','3')= 1;
c('3','4')= 1;
c('3','5')= 1;
c('3','6')= 1;
c('3','7')= 1;
c('3','8')= 1;
c('3','9')= 1;
c('3','10')= 1;

Parameters
b(i,j)            element of transition matrix for graph G;
*  =1 - when the arc number j starts in the i-th node
* =-1 - when the arc number j ends in the i-th node
*  =0 - otherwise;

b('1','1')= 1;
b('1','2')= 1;
b('1','3')= 1;
b('2','1')= -1;
b('2','4')= 1;
b('2','7')= -1;
b('2','8')= 1;
```

```
b('3','2')= -1;
b('3','4')= -1;
b('3','6')= -1;
b('3','5')= 1;
b('3','7')= 1;
b('3','9')= 1;
b('4','3')= -1;
b('4','5')= -1;
b('4','6')= 1;
b('4','10')= 1;
b('5','8')= -1;
b('5','9')= 1;
b('5','10')= 1;

Parameters
c_opt(n)         optimal value of the n-th criteria function;
c_opt('1')= 6;
c_opt('2')= 5;
c_opt('3')= 2;

Parameters
alfa(n)          weight of the n-th criteria function;
alfa('1')= 1/3;
alfa('2')= 1/3;
alfa('3')= 1/3;

Parameters
d(i)             parameter to set source and destination nodes;
* = 1 for source node,
* =-1 for destination node,
* =0 otherwise;

d('1')= 1;
d('2')= 0;
d('3')= 0;
d('4')= 0;
d('5')= -1;

Variables
x(j)
z;

Positive variable  x;

Equations
objective       objective function (3.73)
subj(i)         condition (3.60);

objective.. z =e= sum((n,j), (alfa(n)/c_opt(n))*c(n,j)*x(j));
subj(i).. sum(j,x(j)*b(i,j))=e=d(i);

Model mosp_lp3 /all/ ;

option limrow=16;
*number of rows in output file
```

```
option reslim=10000;
*10000 seconds for calculations;

option iterlim=100000000;
* upper bound on iteration numbers

option lp=Cplex;
* solver Cplex

solve mosp_lp3 using lp minimizing z;
display x.l, z.l;
```

By solving this model using the GAMS/CPLEX 12.2 solver we obtain: x(1)=x(8)=1 (values of x variable for remaining parameters are equal 0), and the value of the objective function equals 1.055 (see also Table 3.9).

3.3.5. Numerical Examples and Analysis

In Fig. 3.12 we present a graph, which will be used as a running example of defined the *MOSP* problems with three-dimensional vector of costs (Tarapata, 2007d). Values of all functions are minimized.

In Table 3.8 we present the set of paths from s_1=1 to t_1=5 for the graph from Fig. 3.12 and their multidimensional properties. In the last row of the table optimal costs for each of the objectives are presented ($c^* = (6,5,2)$). In the last column of Table 3.8 for each of the path I from s_1=1 to t_1=5 the smallest value of (1+ε) such that $F(I) \overset{\varepsilon}{\leq} c^*$ is calculated. For example, for pA we have: 1+ε=max{7/6, 5/5, 2/2}=7/6. Table 3.9 contains optimal multidimensional paths for the graph from Fig. 3.12 (s_1=1, t_1=5) using different types of the defined *MOSP* problems.

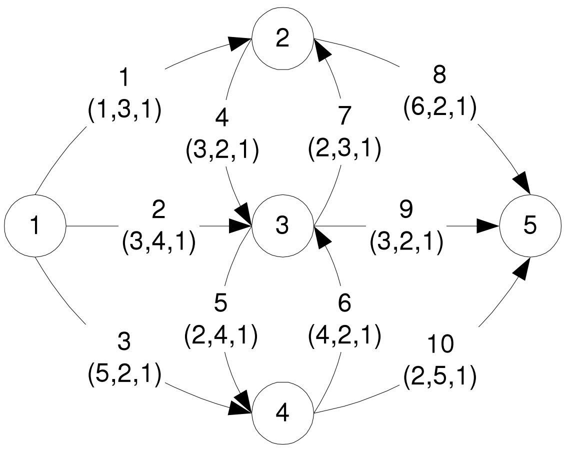


Fig. 3.12. Exemplified graph with multidimensional costs: on top of each arc its number is described and on the bottom – three-component arc cost

Table 3.8. Set of paths from s_1=1 to t_1=5 for the graph from Fig. 3.12 and their multidimensional properties

Path name I	*Path as a sequence of nodes*	*Cost vector F(I) of path*	*1+ε*
pA	1-2-5	(7, 5, 2)	7/6
pB	1-2-3-5	(7, 7, 3)	3/2
pC	1-2-3-4-5	(8, 14, 4)	14/5
pD	1-3-5	(6, 6, 2)	6/5
pE	1-3-2-5	(12, 8, 3)	12/6
pF	1-3-4-5	(7, 9, 3)	9/5
pG	1-4-5	(7, 7, 2)	7/5
pH	1-4-3-5	(12, 6, 3)	12/6
pI	1-4-3-2-5	(17, 9, 4)	17/6
Vector of optimal costs: $c_1^* = 6, c_2^* = 5, c_3^* = 2$			

Table 3.9. Optimal multidimensional paths for the graph from Fig. 3.12 (s_1=1, t_1=5)

Problem	*Optimal path*	*Cost of path*
MF_1, $\alpha_n = 1/3,\ n = \overline{1,3} \Leftrightarrow$ *MOSP_LP3*	*pA*	1.055
MF_1, $\alpha_1 = 0.66, \alpha_2 = 0.17, \alpha_3 = 0.17 \Leftrightarrow$ *MOSP_LP3*	*pD*	1.034
$CS_{p=1} \Leftrightarrow$ *MOSP_LP2*	*pA*	3.167
RSPP⇔*MOSP_LP4*, *L*=1, $g_2 = 1.2c_2^*$, $g_3 = 1.2c_3^*$	*pD*	6.0
RSPP⇔*MOSP_LP4*, *L*=1, $g_2 = 1.1c_2^*$, $g_3 = 1.1c_3^*$	*pA*	7.0
RSPP⇔*MOSP_LP4*, *L*=1, $g_2 = c_2^*$, $g_3 = c_3^*$	null	+infinity
$CS_{p=2} \Leftrightarrow$ *MOSP_NP1*	*pA*	0.139
$CS_{p=\infty} \Leftrightarrow$ *MOSP_NP2*	*pA, pD*	0.333
MOSP_NP3	*pA*	*u*=1/6

In Fig. 3.13, Fig. 3.14 and Fig. 3.15 we present weighted terrain-based grid graphs with a dimension of 50×200 nodes (squares) representing the neighbourhood of Radom, Poland. Each of the graphs has an arc count of $A \approx 3{,}95V$, because only north-east-south-west moves are permitted from a node. Such graphs represent a model of the battlefield in a computer simulation game (Tarapata, 2003a). For this example, each terrain square has a size of 200×200m, so graphs represent a piece of terrain with a dimension of 10×40km. Colours represent values of criteria: c_1 for Fig. 3.13 – the light colour of the node (square) describes open terrain (well passable), the dark colour describes obstacles (forests, lakes, rivers, buildings): the darkest is the colour of least passable terrain; c_2 for Fig. 3.14 – the colour of the node (square) describes ability to camouflage: the darker the colour, the smaller the ability to camouflage; c_3 for Fig. 3.15 – values of criterion c_3 equals 1 for all nodes. The white colour on all figures describes the optimal path from the left-top corner to the right-bottom. Let us note that finding the optimal path in a sense of: c_1 gives the fastest path, c_2 gives the best "camouflaged" path, c_3 gives the shortest geometric path (with north-east-south-west moves only from a given node). Without loss of generality we can assume that functions c_1, c_2, c_3 are described on the nodes (squares) instead of arcs (if it is necessary to obtain a graph with arc functions we can construct a dual graph $GT = \langle V_{GT}, A_{GT} \rangle$ to the considered graph $G = \langle V_G, A_G \rangle$, where $V_{GT} = A_G$ and each arc $(a,b) \in A_{GT} \subset A_G \times A_G$ is created when two arcs a, b in G have a common node (are simultaneously adjacent with any node); then in GT functions c_1, c_2, c_3 are described on arcs).

In Table 3.10 we present experimental results of average running times (in seconds) of the modified Dijkstra's algorithm and the GAMS/CPLEX 12.2 for the MF_1 problem ($\alpha_i=1/N$, $i=1,\dots,N$). Graphs with a node count of 1000*x ($x=1,2,\dots,10$) are cut from the graph with 50×200 nodes (Fig. 3.13, Fig. 3.14, Fig. 3.15) and have a dimension of 50×(20*x) nodes.

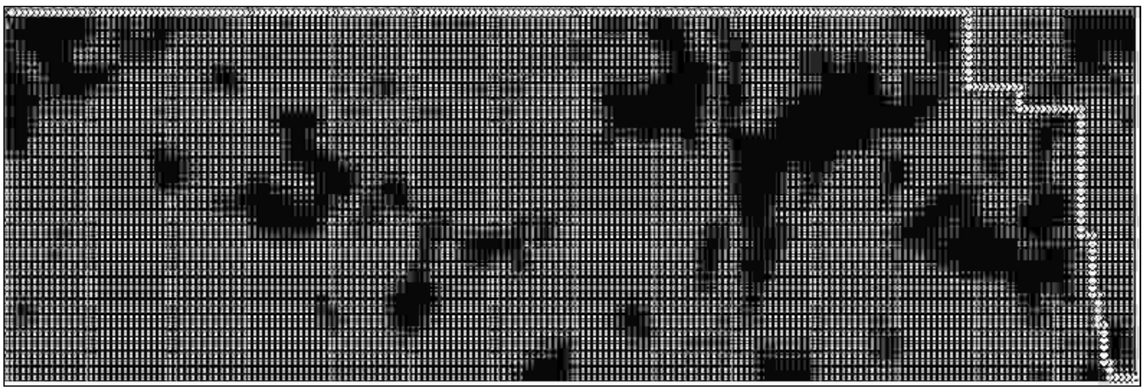


Fig. 3.13. Weighted terrain-based grid graph with a dimension of 50x200 nodes (squares). Colour represents value of criterion c_1: the light colour of the nodes (square) describe open terrain, the dark colour describes obstacles (forests, lakes, rivers, buildings). The white colour describes the optimal path from the left-top to the right-bottom corner

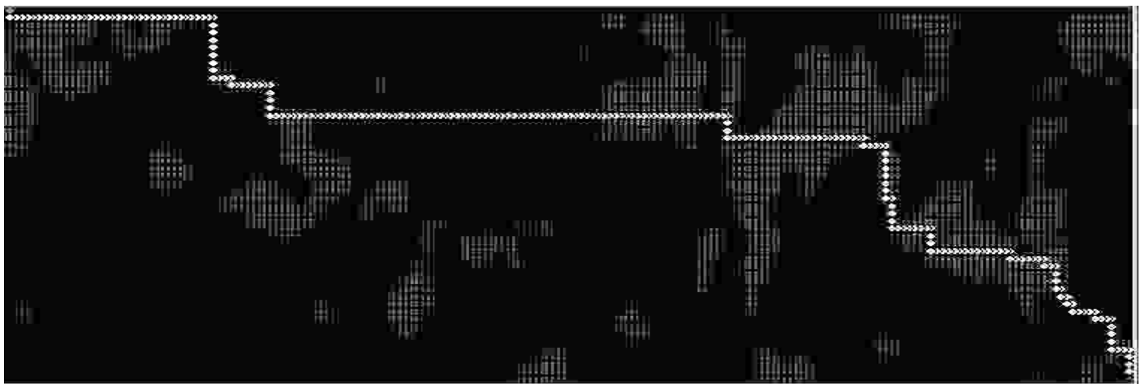


Fig. 3.14. Weighted terrain-based grid graph with a dimension of 50x200 nodes (squares). Colour represents value of criterion c_2: the colour of the node (square) describes the ability to camouflage: the darker the colour the smaller the ability to camouflage. The white colour describes the optimal path from the left-top to the right-bottom corner

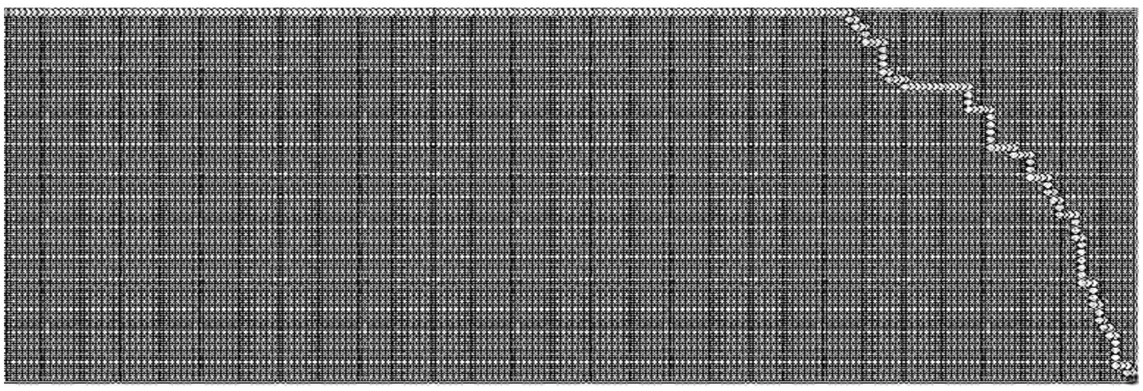


Fig. 3.15. Weighted terrain-based grid graph with a dimension of 50x200 nodes (squares). All weights are identical (the c_3 criterion value equals 1). The white colour describes the optimal path from the left-top to the right-bottom corner

We can see a clear advantage of the modified Dijkstra's algorithm with relation to CPLEX 12.2 solving the *MF_1* problem as a linear programming problem *MOSP_LP*3: using the modified Dijkstra's algorithm with its fast implementations is time-effective. It is especially visible in Fig. 3.16 where we present a decimal logarithm of the average running times (in milliseconds) of these two algorithms.

Table 3.10. Average running times (in seconds) of the modified Dijkstra's algorithm and the GAMS/CPLEX 12.2 solver for the *MF_1* problem (α_i=1/*N*, *i*=1,…,*N*)

Count of nodes (V)	*Modified Dijkstra's alg.*			*MF_1 solved as MOSP_LP3*		
	N=1	*N=2*	*N=3*	*N=1*	*N=2*	*N=3*
1 000	0.03	0.08	0.11	0.76	2.31	4.39
2 000	0.10	0.29	0.38	2.82	8.81	12.40
3 000	0.25	0.71	0.96	6.52	21.20	29.14
4 000	0.37	1.10	1.47	16.40	52.55	72.30
5 000	0.59	1.74	2.33	30.41	98.12	136.22
6 000	0.86	2.55	3.42	50.79	161.94	225.67
7 000	1.16	3.44	4.59	74.61	238.27	333.80
8 000	1.55	4.57	6.12	109.24	348.13	483.76
9 000	1.96	5.82	7.77	134.78	432.47	620.94
10 000	2.43	7.24	9.66	179.61	564.42	790.97

Fig. 3.17 presents dependencies between the average running times (in milliseconds) of the GAMS/CPLEX 12.2 solver and the *beta* coefficient for solving the *MOSP_LP*4 problem for two graphs with *V*=1 000 (50×20) and *V*=2 000 (50×40) nodes. In the *MOSP_LP*4 problem we minimize the c_1 criterion subject to upper constraints (g_2 and g_3) on values of criteria c_2 and c_3 as follows: $g_2 = beta \cdot c_2^*$ and (g_3=infinity, $g_3 = beta \cdot c_3^*$), where $c_2^* = 6\,964$ and $c_3^* = 68$ for *V*=1 000; $c_2^* = 6\,061$ and $c_3^* = 88$ for *V*=2 000.

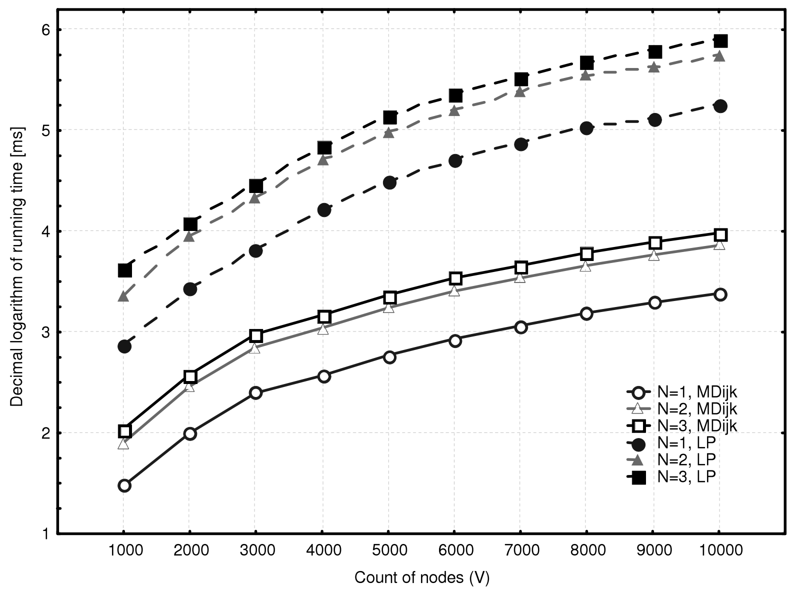


Fig. 3.16. Decimal logarithm of average running times (in milliseconds) of the modified Dijkstra's algorithm (*MF*_1 problem⇔MDijk) and the GAMS/CPLEX 12.2 (*MOSP_LP*3 problem⇔LP) (α_i=1/*N*, *i*=1,…,*N*)

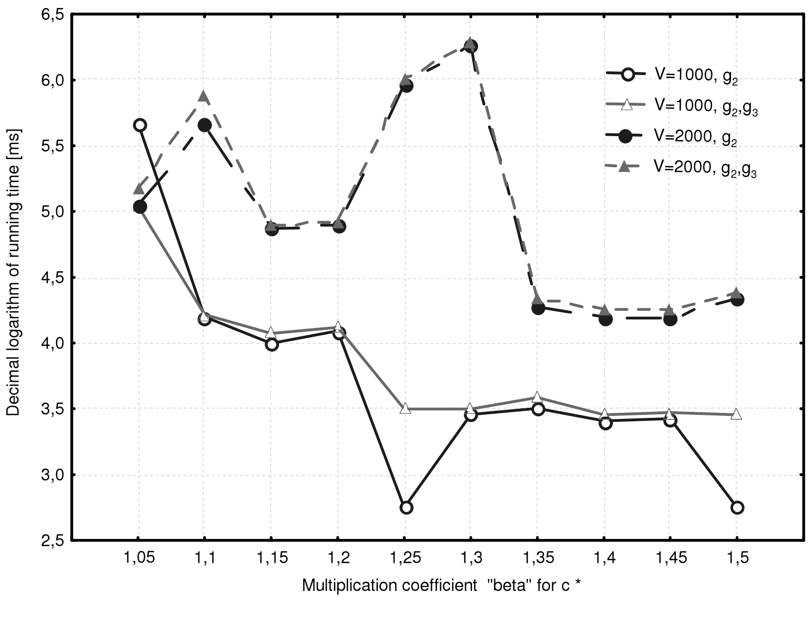


Fig. 3.17. Decimal logarithm of average running times (in milliseconds) of the GAMS/CPLEX 12.2 solver solving the *MOSP_LP*4 problem for two graphs with *V*=1 000 and *V*=2 000 nodes, $g_2 = beta \cdot c_2^*$ and (g_3 = infinity , $g_3 = beta \cdot c_3^*$)

In Fig. 3.18 we present dependencies between values of the objective function and *beta* coefficient for the *MOSP_LP*4 problem. Let us note that, generally, the greater the value of *beta* the smaller running time of the model in the GAMS/CPLEX 12.2 solver (and the smaller value of the objective functions, Fig. 3.18), but the functions from Fig. 3.17 are not monotonic. The values of the running times for the *MOSP_LP*4 problem are few times greater than for the *MOSP_LP*3 problem solved using the GAMS/CPLEX 12.2 solver (compare Fig. 3.17 and Fig. 3.16). For example, the running time for *V*=2 000 is about $10^5/10^{2.8}$ times greater than for solving the *MOSP_LP*3 problem. These results are clear: the smaller restrictions on criteria c_2 and c_3 (that means: the greater value of *beta*) the smaller running time. Moreover, the greater value of the running time results from the fact that $g_i = beta \cdot c_i^*$ is not an integer (except for *beta*=1.25 and *beta*=1.5 for $c_2^* = 6\,964$, *V*=1 000) and *MOSP_LP*4 (as a linear programming problem) becomes harder to solve the binary programming problem. For the *beta*≥1.35 value of the objective function (based on c_1) does not change, because it achieves an optimal value ($c_1^* = 605$ for *V*=1 000, $c_1^* = 713$ for *V*=2 000).

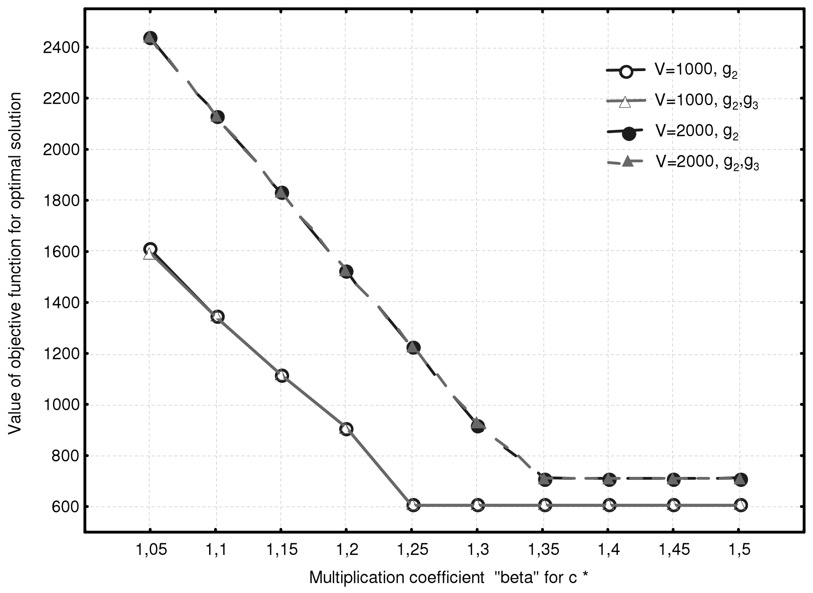


Fig. 3.18. Values of objective functions for the *MOSP_LP*4 problem for two graphs with *V*=1 000 and *V*=2000 nodes, $g_2 = beta \cdot c_2^*$ and ($g_3 = \text{infinity}$, $g_3 = beta \cdot c_3^*$)

3.4. Disjoint Paths Planning (*DP*)

3.4.1. Description of the Problem

The disjoint paths (*DP*) problem is a well-known network optimization problem. The problem relies on such determining paths for a few objects that no common part of paths for objects (arcs or nodes belonging to paths) are accepted. There are two classification categories of the problem: (*DP*1) from the point of view of paths disjointness type; (*DP*2) from the point of view of source and destination type. In the (*DP*1) category the problem is divided into two subproblems: (*DP*1.1) arc-disjoint paths (no common arcs are accepted) and (*DP*1.2) node-disjoint paths (no common nodes are accepted; in selected cases, common source and destination nodes can be accepted). It is easy to note that the following sentence is true: if two (or more) paths are node-disjoint then they are arc-disjoint as well. Reverse relation can not be true. In the (*DP*2) category the problem is divided into a few subproblems: (*DP*2.1) from a single source to a single destination; (*DP*2.2) from a single source to a set of destination ones; (*DP*2.3) from a set of sources to a single destination; (*DP*2.4) from a set of sources to a set of destinations; (*DP*2.5) from a vector of sources to a vector of destinations. What is the difference between these models? When we have *K*-component vectors of sources ($s = \langle s_1, s_2, ..., s_K \rangle$) and destinations ($t = \langle t_1, t_2, ..., t_K \rangle$) then we must find disjoint paths from s_1 to t_1 and from s_2 to t_2, and from s_3 to t_3, etc., hence the *K* disjoint paths between *K* **pairs** of nodes. When we have *K*-element sets of sources and destinations we must find *K* disjoint paths between **any** of the sources and destinations. In general, the case with the vector of sources and/or destinations is more complicated to solve then with a set of them.

The disjoint paths problem may be related to the following practical applications: VLSI layout designing (Aggarwal *et al.*, 2000), routing in telecommunication networks (in particular: optical) (Aggarwal *et al.*, 2000; Ahuja *et al.*, 1993; Andersen *et al.*, 2004; Bhandari, 1999; Jongh *et al.*, 1999; Li *et al.*, 1992; Perl & Shiloach; 1978, etc.), manoeuvre (transport) planning of military detachments (or vehicles) (Tarapata, 1998; 2008e; 2009a; Tarapata & Wroclawski, 2010g; 2011d), tasks scheduling (trasmission) in a parallel or a distributed computing system (Tarapata, 1999a; 2000e), couriers problem (Tarapata, 1998). For example, in military applications, to increase redeployment safety, it is often required that paths for moved objects (convoys) should be independent (disjoint). These disjoint paths condition results from the fact, that during convoy redeployment the potential opponent may try to destroy structure elements of the network (for example, crossings (node of the network) or parts of the road, bridges (arcs of the network)) as well as convoys being redeployed to make impossible the achievement destinations and intended goals by the convoys.

It is known (Even *et al.*, 1976; Perl & Shiloach; 1978) that the optimization problem for finding K>1 shortest disjoint paths between *K* pairs of distinct nodes (*DP*2.5 problem) is NP-hard (even for K=2). The problem of finding two or more of disjoint paths between specified pairs of terminals (network nodes) has been well studied. The first significant result in this subject has been presented in (Suurballe, 1974). Presented in this paper is the algorithm for the single source – single destination case having a complexity of $O(A\log_{(1+A/N)}V)$, where V – number of network nodes, A – number of network arcs. This procedure solved the problem as a special case of a minimum-cost network flow problem using two efficient implementations of the Dijkstra's single–source shortest path algorithm. An efficient algorithm to solve the problem for the single–source all destinations node–disjoint paths was given in (Suurballe & Tarjan, 1984). In this study, the disjoint pairs of paths from the source node to all the other nodes in the network are determined using a single Dijkstra–like calculation to derive an algorithm having a time complexity of $O(A\log_{(1+A/V)}V)$. Perl and Shiloach (Perl & Shiloach; 1978) studied the complexity of finding two disjoint paths between two different sources and two different destinations in directed acyclic graphs (*DAGs*). They proposed an algorithm, which is easily generalized in finding the shortest pair of paths (measured by the total path length) or finding tuples of d disjoint paths between distinct specified terminals; in the latter case the running time would become $O(AV^{d-1})$. The author of the paper (Eppstein, 1995) considered the problem of finding pairs of node-disjoint paths in *DAGs*, either connecting two given nodes to a common ancestor, or connecting two given pairs of terminals. He showed how to find the K pairs with the shortest combined length in a time of $O(AV+K)$. He also showed how to count all such pairs of paths in $O(AV)$ arithmetic operations. These results can be extended to finding or counting tuples of d disjoint paths in a time of $O(AV^{d-1}+K)$ or $O(AV^{d-1})$. Authors of the paper (Li *et al.*, 1990) give a pseudo-polynomial algorithm for an optimization version of the two-path problem, in which the length of the longer path must be minimized. In the other paper of these authors (Li *et al.*, 1992) the difficult bifurcated routing problem was described. They solved the problem when each path corresponds to the routing of a different commodity so that each arc is endowed with a cost depending on the path to which it belongs. In the paper (Jongh *et al.*, 1999) the problem of finding two node disjoint paths with minimum total cost in the network was studied, in which a cost is associated with each arc or edge and a transition cost is associated with each node. This last cost is related to the presence of two technologies on the network and is incurred only when a flow enters and leaves the corresponding node or arcs of different types. A good study for a very important problem of finding disjoint paths in planar graphs was presented in paper (Schrijver & Seymour, 1992). A very interesting approach to the time-dependent shortest pair of disjoint paths problem was discussed in (Sherali *et al.*, 1998). In (Tarapata, 1997; 1998; 1999a; 2000e) a new approach to the K disjoint path problem was proposed: it is based on building, starting from the initial network, the so-called K-nodes (K-dimensional vectors of network nodes), K-arcs and "virtual" K-network, and finding in such a K-network the shortest K-path (K-dimensional vector of simple paths) using the original Dijkstra-like algorithm. The specific problem has been considered in the papers. It deals with the parallel or distributed computing system, in which we want to send (or process), in generality, K (K>1) tasks from the K^s (K^s=1 or K^s=K) computer-nodes (local servers) to the K^d (K^d=1 or K^d=K) destination ones through disjoint paths to minimize sending (or processing) the time of all tasks and simultaneously to ensure task sending (or processing) on the most reliable paths (when the elements of the network structure are unreliable). One of the methods being proposed to solve the problem is finding the best paths for *K* objects iteratively using methods for finding the *m*-th (1st, 2nd, etc.) best path for each of the *K* objects (Eppstein, 1999) and visiting specified nodes (Ibaraki, 1973). Parallelization of the method is presented in (Tarapata, 2000a).

3.4.2. Definition of the Problem

3.4.2.1. Formulation of the node-disjoint paths visiting specified nodes problem

The mesh graph, which is the basic data for the problem can model, for example, regular grid of terrain squares used to plan off-road (cross-country) movement (see Fig.2.3b). This grid divides the terrain space into squares of equal size. Each square is homogeneous from the point of view of terrain characteristics (dimensions, degree of slowing down the velocity, ability to camouflage, degree of visibility, etc.). The structure of such a terrain can be represented by a "mesh" digraph $G=\langle V_G, A_G\rangle$, V_G – set of graph nodes (V_G describes the centre of terrain squares), A_G – set of graph arcs, $A_G \subset V_G \times V_G$, $A=\overline{\overline{A_G}}$. Arcs are allowed between geographically adjacent squares only (see Fig.2.3b).

To define the considered problem let us accept the following descriptions: $s=\langle s_1, s_2, ..., s_K\rangle$ – vector of source nodes, $t=\langle t_1, t_2, ..., t_K\rangle$ – vector of destination nodes, $\mathbf{A}=[a_{ink}]_{V\times M\times K}$ – matrix of source and destination nodes via indirect nodes for each object (a path for each object is divided into $M=N+1$ parts (segments) from one node to other indirect nodes, N – number of indirect nodes): $a_{ink}=1$ if the i-th node is the n-th source node for the k-th object, $a_{ink}=-1$ if the i-th node is the n-th destination node for the k-th object; $a_{ink}=0$ otherwise; additionally, the following conditions must be satisfied: $a_{i1k}=1 \Leftrightarrow i=s_k$ (it means that node s_k must be the source node of the first segment of the path for the k-th object), $a_{i1k}=-1 \Leftrightarrow i=i_1(k)$ (the first indirect node $i_1(k)$ for the k-th object is the destination node for the first segment of the path for this object), $a_{iMk}=1 \Leftrightarrow i=i_N(k)$ (the last indirect node $i_N(k)$ for the k-th object is the source node for the last segment of the path for this object), $a_{iMk}=-1 \Leftrightarrow i=t_k$, (node t_k is the destination node of the last segment of the path for the k-th object), $\underset{n\in\{1,...,N\}}{\forall} a_{ink}=-1 \Rightarrow a_{i(n+1)k}=1$ (the destination node of the n-th path segment for the k-th object is, simultaneously, the source node of (n+1)-st segment for this path); $\mathbf{H}=[h_{ik}]_{V\times K}$ – matrix of nodes (generating subgraphs of G), which are allowed to be taken into account during paths determination for each object: $h_{ik}=1$ if the i-th node can be taken into account during paths determining for the k-th object, $h_{ik}=0$ - otherwise (in particular: $i=s_k \Rightarrow h_{ik}=1$, $i=t_k \Rightarrow h_{ik}=1$); $\mathbf{OUT}=\left[out_{ij}\right]_{V\times A}$ –

binary crossing matrix of arcs starting in nodes of *G*: out_{ij}=1 if the *j*-th arc starts in the *i*-th node, out_{ij}=0 - otherwise; $\mathbf{IN} = \left[in_{ij} \right]_{V \times A}$ – binary crossing matrix of arcs ending in nodes of *G*: in_{ij}=1 if the *j*-th arc ends in the *i*-th node, in_{ij}=0 – otherwise; $\mathbf{D} = \left[d_j \right]_{1 \times A}$ – vector of arcs' cost; $\mathbf{X} = \left[x_{jnk} \right]_{A \times M \times K}$ – decision variables matrix, x_{jnk}=1 if the *j*-th arc of *G* belongs to the *n*-th segment of the path for the *k*-th object, otherwise x_{jnk}=0 .

We can formulate two problems (*NDRP-Sum* and *NDRP-Max,* both are a modification of the *DP*2.5 problem), which differ in the objective function. The first one (*NDRP-Sum*) minimizes the total cost of all (*K*) disjoint paths visiting specified nodes in the restricted area and the second one (*NDRP-Max*) minimizes the maximal cost of any of the *K* disjoint paths.

The optimization *NDRP-Sum* problem of determining the *K* shortest node-disjoint paths via some indirect nodes in the restricted area can be defined as follows:

$$\sum_{j=1}^{A} \sum_{n=1}^{M} \sum_{k=1}^{K} d_j x_{jnk} \rightarrow \min \tag{3.78}$$

with constraints:

$$\sum_{j=1}^{A} \left(out_{ij} - in_{ij} \right) x_{jnk} = a_{ink}, \qquad i = \overline{1,V},\; n = \overline{1,M},\; k = \overline{1,K} \tag{3.79}$$

$$\sum_{j=1}^{A} \sum_{n=1}^{M} \sum_{k=1}^{K} out_{ij} x_{jnk} \le 1, \qquad i = \overline{1,V} \tag{3.80}$$

$$\sum_{j=1}^{A} \sum_{n=1}^{M} \sum_{k=1}^{K} in_{ij} x_{jnk} \le 1, \qquad i = \overline{1,V} \tag{3.81}$$

$$\sum_{j=1}^{A} \sum_{n=1}^{M} out_{ij} x_{jnk} \le h_{ik}, \qquad i = \overline{1,V},\; k = \overline{1,K} \tag{3.82}$$

$$\sum_{j=1}^{A} \sum_{n=1}^{M} in_{ij} x_{jnk} \le h_{ik}, \qquad i = \overline{1,V},\; k = \overline{1,K} \tag{3.83}$$

$$x_{jnk} \ge 0, \qquad j = \overline{1,A},\; n = \overline{1,M},\; k = \overline{1,K} \tag{3.84}$$

The objective function (3.78) describes the total cost of *K* disjoint paths, which is minimized. The first constraint (3.79) assures that for each node (excluding the source and destination nodes), for each object and for each path segment, the sum of arcs starting from the node and the sum of arcs ending at the node, which are selected to the path is the same (further constraints assure that this value is ≤1). For the source node this difference is equal 1 (only the single path segment can start at the source node) and for the destination node -1 (only the single path segment can end at the destination node). Constraints (3.80) and (3.81) supplement constraint

(3.79) to assure that for each node only, at least one arc starting and ending at that node can belong to any path. Constraints (3.82) and (3.83) guarantee that only allowed nodes are on the path for the *k*-th object (definition of the restricted area). Additionally, it can be observed that the matrix of constraint coefficients (built on the basis of the left sides of the constraints (3.79)-(3.83)) is totally unimodular (proof of this property is presented in (Tarapata, 2007a)) and a_{ink}, h_{ik} (right sides) are integer, hence we can obtain the continuous linear programming problem (instead of the binary linear programming) and constraint (3.84) (instead $x_{jnk} \in \{0,1\}$). In the presented optimization problem we have the *AMK* decision variables and $V(MK+K+2)$ constraints (excluding (3.84)). Let us note that we could use transition matrix $B=\left[b_{ij}\right]_{V\times A}$ of graph *G* (defined in chapter 3.3.4.6) instead of the semi-transition matrices **OUT** and **IN**. In such a case we could have the following constraints:

(3.79) => $\sum_{j=1}^{A} b_{ij}x_{jnk} = a_{ink}$, (3.80) => $\sum_{j=1}^{A}\sum_{n=1}^{M}\sum_{k=1}^{K} b_{ij}x_{jnk} \le 1$, (3.81) => $\sum_{j=1}^{A}\sum_{n=1}^{M}\sum_{k=1}^{K} b_{ij}x_{jnk} \ge -1$,

(3.82) => $\sum_{j=1}^{A}\sum_{n=1}^{M}\sum_{k=1}^{K} b_{ij}x_{jnk} \le h_{ik}$, (3.83) => $\sum_{j=1}^{A}\sum_{n=1}^{M}\sum_{k=1}^{K} b_{ij}x_{jnk} \ge -h_{ik}$.

Matrices **OUT** and **IN** have been used because of computational reasons without increasing computational complexity of the problem.

The *NDRP-Max* problem can be formulated similarly to the *NDRP-Sum* problem, excluding the objective function, which has a form:

$$\max_{k\in\{1,\ldots,K\}} \sum_{j=1}^{A}\sum_{n=1}^{M} d_j x_{jnk} \to \min \tag{3.85}$$

and with constraints (3.79)-(3.84).

Unfortunately, the function (3.85) is nonlinear and the *NDRP-Max* problem is nonlinear. We can use the equivalent formulation of the problem to avoid its nonlinearity:

$$u \to \min \tag{3.86}$$

with constraints:

$$\sum_{j=1}^{A}\sum_{n=1}^{M} d_j x_{jnk} \le u, \qquad k=\overline{1,K} \tag{3.87}$$

and (3.79)-(3.84).

Formulation (3.86)-(3.87) of the *NDRP-Max* problem makes it a linear programming problem.

3.4.2.2. Example of the GAMS model for the *NDRP-Sum* problem

Below we present the GAMS model for the *K*=2 *NDRP-Sum* problem from *s*=(1,2) to *t*=(7,8) in graph *G* from Fig. 3.19.

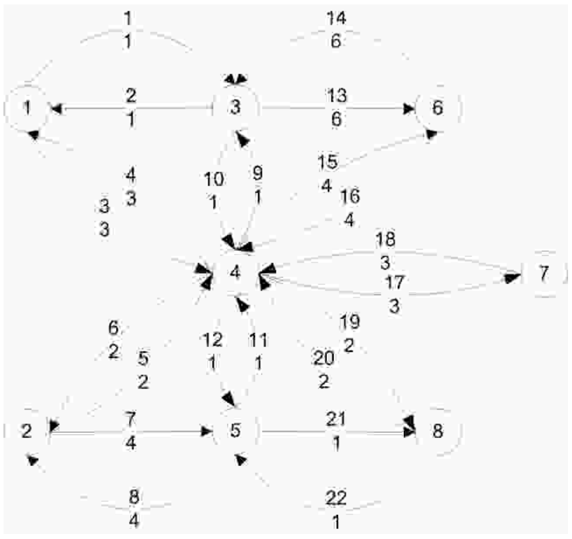


Fig. 3.19. Graph *G* for the *K*=2 disjoint paths GAMS model: on top of each arc its number is described and on the bottom – arc cost

We set the following equivalence between notations being used in model of the *NDRP-Sum* problem (defined by (3.78)-(3.84)) and in the source code of the GAMS model (notation *x*≡*y* describes that *x* in the GAMS model is equivalent to *y* in the *NDRP-Sum* model): d(j) $\equiv d_j$, out(i,j) $\equiv out_{ij}$, in(i,j) $\equiv in_{ij}$, a(i,n,k) $\equiv a_{ink}$, x(j,n,k) $\equiv x_{jnk}$.

```
Sets
i               set of nodes of graph G
/1, 2, 3, 4, 5, 6, 7, 8/

j               set of arcs of graph G
/1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
19, 20, 21, 22/

k              set of objects (paths)
/1, 2/

n              set of indirect nodes
/1, 2/

Parameters
d(j)                arcs cost vector;
 d('1')= 1 ;
 d('2')= 1;
 d('3')= 3 ;
 d('4')= 3 ;
 d('5')= 4 ;
 d('6')= 4 ;
 d('7')= 2 ;
 d('8')= 2 ;
 d('9')= 1 ;
 d('10')= 1;
 d('11')= 1 ;
 d('12')= 1 ;
 d('13')= 6 ;
 d('14')= 6 ;
 d('15')= 4 ;
 d('16')= 4 ;
 d('17')= 3 ;
 d('18')= 3 ;
 d('19')= 2 ;
 d('20')= 2;
 d('21')= 1;
 d('22')= 1;

Parameters
out(i,j)   binary crossing matrix of arcs starting in nodes of G;
*  1 - if the j-th arc starts in the i-th node
*  0 - otherwise;

out('1','1')= 1 ;
out('1','3')= 1 ;
out('2','5')= 1 ;
out('2','7')= 1 ;
out('3','2')= 1 ;
out('3','10')= 1 ;
out('3','13')= 1 ;
out('4','9')= 1 ;
out('4','15')= 1 ;
out('4','17')= 1 ;
out('4','19')= 1 ;
out('4','12')= 1 ;
out('4','6')= 1 ;
out('4','4')= 1 ;
out('5','9')= 1 ;
out('5','11')= 1 ;
out('5','21')= 1 ;
out('6','14')= 1 ;
out('6','16')= 1 ;
out('7','18')= 1 ;
out('8','20')= 1 ;
out('8','22')= 1 ;

Parameters
in(i,j)      binary crossing matrix of arcs ending in nodes of G;
*  1 - if the j-th arc ends in the i-th node
*  0 - otherwise;

in('1','2')= 1 ;
in('1','4')= 1 ;
in('2','6')= 1 ;
in('2','8')= 1 ;
in('3','1')= 1 ;
in('3','9')= 1 ;
in('3','14')= 1 ;
in('4','3')= 1 ;
in('4','5')= 1 ;
in('4','11')= 1 ;
in('4','20')= 1 ;
in('4','18')= 1 ;
in('4','16')= 1 ;
in('4','10')= 1 ;
in('5','7')= 1 ;
in('5','12')= 1 ;
in('5','22')= 1 ;
in('6','13')= 1 ;
in('6','15')= 1 ;
in('7','17')= 1 ;
in('8','19')= 1 ;
in('8','21')= 1 ;

Parameters
a(i,n,k)      source and destination nodes via indirect nodes;
*a(i,n,k)=1   if the i-th node is the n-th source node for the
              k-th object,
*a(i,n,k)=-1  if the i-th node is the n-th destination node for
              the k-th object,
*a(i,n,k)=0   otherwise;

a('1','1','1')=1;
a('7','1','1')=-1;
a('2','1','2')=1;
a('8','1','2')=-1;

Variables
x(j,n,k)
z;

Positive variable  x;

Equations
cost                 total paths cost (eq. (3.78))
constr1(i,n,k)       eq. (3.79)
constr2(i)           eq. (3.80)
constr3(i)           eq. (3.81);

cost.. z =e= sum((j,n,k), d(j)*x(j,n,k));
constr1 (i,n,k).. sum(j, (out(i,j)-in(i,j))*x(j,n,k))=e=a(i,n,k);
constr2 (i)..sum((j,n,k), out(i,j)*x(j,n,k))=l=1;
constr3 (i)..sum((j,n,k), in(i,j)*x(j,n,k))=l=1;

Model DisjPathsSum /all/;

option limrow=16;
*number of rows in output file

option reslim=10000;
*10000 seconds for calculations;

option iterlim=100000000;
```

```
* upper bound on iteration numbers

option lp=Cplex;
* solver Cplex

solve DisjPathsSum using lp minimizing z;
display x.l, z.l;
```

By solving this model using the GAMS/CPLEX 12.2 solver we obtain:

```
----    154 VARIABLE x.L
               1           2
1 .1       1.000
7 .1                   1.000
10.1       1.000
17.1       1.000
21.1                   1.000

----    154 VARIABLE z.L                     =       10.000
```

It means that for the 1st object we have obtained the path (as a sequence of arcs): 1-10-17 and for the 2nd one: 7-21. Total cost of this *K*=2 node-disjoint paths =10.

3.4.2.3. Example of the GAMS model for the *NDRP-Max* problem

Below we present the GAMS model for the *K*=2 *NDRP-Max* problem from *s*=(1,2) to *t*=(7,8) in graph *G* from Fig. 3.19. We set the following equivalence between notations being used in the model of the *NDRP-Max* problem (defined by (3.78)-(3.84) and (3.86), (3.87)) and in the source code of the GAMS model (notation *x*≡*y* describes that *x* in the GAMS model is equivalent to *y* in the *NDRP-Max* model): $\mathrm{d(j)} \equiv d_j$, $\mathrm{out(i,j)} \equiv out_{ij}$, $\mathrm{in(i,j)} \equiv in_{ij}$, $\mathrm{a(i,n,k)} \equiv a_{ink}$, $\mathrm{x(j,n,k)} \equiv x_{jnk}$.

```
Sets
i              set of nodes of graph G
/1, 2, 3, 4, 5, 6, 7, 8/

j              set of arcs of graph G
/1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
19, 20, 21, 22/

k             set of objects (paths)
/1, 2/

n              set of indirect nodes
/1, 2/

Parameters
d(j)            arcs cost vector;
 d('1')= 1 ;
 d('2')= 1;
 d('3')= 3 ;
 d('4')= 3 ;
 d('5')= 4 ;
 d('6')= 4 ;
 d('7')= 2 ;
 d('8')= 2 ;
 d('9')= 1 ;
 d('10')= 1;
 d('11')= 1 ;
 d('12')= 1 ;
 d('13')= 6 ;
 d('14')= 6 ;
 d('15')= 4 ;
 d('16')= 4 ;
 d('17')= 3 ;
 d('18')= 3 ;
 d('19')= 2 ;
 d('20')= 2;
 d('21')= 1;
 d('22')= 1;

Parameters
out(i,j)   binary crossing matrix of arcs starting in nodes of G;
*  1 - if the j-th arc starts in the i-th node
*  0 - otherwise;

out('1','1')= 1 ;
out('1','3')= 1 ;
out('2','5')= 1 ;
out('2','7')= 1 ;
out('3','2')= 1 ;
out('3','10')= 1 ;
out('3','13')= 1 ;
out('4','9')= 1 ;
out('4','15')= 1 ;
out('4','17')= 1 ;
out('4','19')= 1 ;
out('4','12')= 1 ;
out('4','6')= 1 ;
out('4','4')= 1 ;
out('5','9')= 1 ;
out('5','11')= 1 ;
out('5','21')= 1 ;
out('6','14')= 1 ;
out('6','16')= 1 ;
out('7','18')= 1 ;
out('8','20')= 1 ;
out('8','22')= 1 ;

Parameters
in(i,j)      binary crossing matrix of arcs ending in nodes of G;
*  1 - if the j-th arc ends in the i-th node
*  0 - otherwise;

in('1','2')= 1 ;
in('1','4')= 1 ;
in('2','6')= 1 ;
in('2','8')= 1 ;
in('3','1')= 1 ;
in('3','9')= 1 ;
in('3','14')= 1 ;
in('4','3')= 1 ;
in('4','5')= 1 ;
in('4','11')= 1 ;
in('4','20')= 1 ;
in('4','18')= 1 ;
in('4','16')= 1 ;
in('4','10')= 1 ;
in('5','7')= 1 ;
in('5','12')= 1 ;
in('5','22')= 1 ;
in('6','13')= 1 ;
in('6','15')= 1 ;
in('7','17')= 1 ;
in('8','19')= 1 ;
in('8','21')= 1 ;

Parameters
a(i,n,k)      source and destination nodes via indirect nodes;
*a(i,n,k)=1   if the i-th node is the n-th source node for the
             k-th object,
*a(i,n,k)=-1  if the i-th node is the n-th destination node for
             the k-th object,
*a(i,n,k)=0   otherwise;

a('1','1','1')=1;
a('7','1','1')=-1;
a('2','1','2')=1;
a('8','1','2')=-1;

Variables
x(j,n,k)
z;

Positive variable  x, u;

Equations
cost                 maximal paths cost (eq. (3.86))
constr1(i,n,k)       eq. (3.79)
constr2(i)           eq. (3.80)
constr3(i)           eq. (3.81)
constr4(k)           eq. (3.87)

cost.. z =e= u;
constr1 (i,n,k).. sum(j, (out(i,j)-in(i,j))*x(j,n,k))=e=a(i,n,k);
constr2 (i)..sum((j,n,k), out(i,j)*x(j,n,k))=l=1;
constr3 (i)..sum((j,n,k), in(i,j)*x(j,n,k))=l=1;
constr4(k)..sum((j,n), d(j)*x(j,n,k))=l=u;

Model DisjPathsMax /all/;

option limrow=16;
*number of rows in output file
```

```
option reslim=10000;
*10000 seconds for calculations;

option iterlim=100000000;
* upper bound on iteration numbers

option lp=Cplex;
* solver Cplex

solve DisjPathsMax using lp minimizing z;
display x.l, z.l;
```

Solving this model using the GAMS/CPLEX 12.2 solver we obtain:

```
----    156 VARIABLE x.L
                  1           2
1 .1       1.000
7 .1                     1.000
10.1       1.000
17.1       1.000
21.1                     1.000
----    156 VARIABLE z.L                    =        5.000
```

It means that for the 1st object we have obtained path (as a sequence of arcs): 1-10-17 and for the 2nd one: 7-21. Maximal cost of any of *K*=2 node-disjoint paths is equal 5 and is minimal among other *K*=2 node-disjoint paths in the graph *G*.

3.4.3. Description of Algorithms for Solving *DP* Problems

3.4.3.1. Subgraphs-generating node-disjoint paths algorithm (*SGDP*)

For solving the *NDRP-Sum* and *NDRP-Max* problems we propose the subgraphs generating-based algorithm (*SGDP*), (Tarapata, 2001; Tarapata & Wroclawski, 2010g). The algorithm searches for a bundle of node-disjoint paths for the *K* objects, each path consists of 2 or more indirect nodes (including the source and destination). The idea of the algorithm is to generate subgraphs (see Fig. 3.20) in the network of terrain squares (for each moved object we generate a separate subgraph) and afterwards, in each of the subgraphs the Dijkstra's shortest path algorithm is run. Each of these subgraphs is created as follows. We link nodes: source and destination for the given object (if we have, for example, 4 indirect nodes we set the following pairs source-destination: 1-2, 2-3, 3-4) and afterwards we "mark" the right and left from the line linking these node stripes with a width of 0,5sw_k, where sw_k describes the width of the stripe, in which the object should move. Nodes of graph $G=\langle V_G, A_G\rangle$, which centre coordinates are located at this stripe generate the subgraph. It means that the PG_k subgraph for the *k*-th object is defined as follows:

$$PG_k=\langle V_{Gk}, A_{Gk}\rangle \tag{3.88}$$

$$V_{Gk} = \left\{ \begin{array}{c} v \in V_G : x(s_k) + \tan g_k \cdot (y(v) - y(s_k)) - \dfrac{sw_k}{\cos g_k} \le x(v) \le \\ \le x(s_k) + \tan g_k \cdot (y(v) - y(s_k)) + \dfrac{sw_k}{\cos g_k} \end{array} \right\} \tag{3.89}$$

where V_{Gk} – set of subgraph nodes for the k-th object, s_k denotes the source node for the k-th object, and $x(v), y(v)$ - coordinates of the v-th node; A_{Gk} – the set of the subgraph's arcs, $A_{Gk} = \{(v,v') \in V_{Gk} \times V_{Gk} : (v,v') \in A_G\}$.

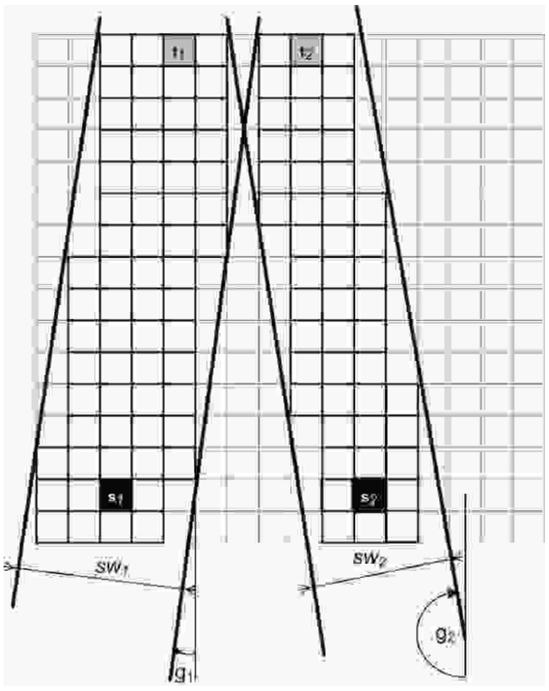


Fig. 3.20. The idea of "cutting" the subgraphs (in the mesh graph) into stripes with a width of sw_k for two moved objects with no indirect nodes

It is possible to exclude some arcs during paths searching by using the passability threshold parameter, which is introduced to reject each arc with cost greater than the given parameter value. Having the subgraph generated for each object we can determine the shortest path for each one in the network based on this subgraph using a few searching strategies. Three strategies are being used to generate the order of objects, for which we find paths: *stripeOrderStrategies*={*Ascending*, *Descending*, *Random*}. The first two strategies are based on order of requests: *Ascending* – order is the same as in the given paths to find; *Descending* – the order is reversed. In *Random* the strategy generated order is randomized with a uniform distribution. By searching with a nondeterministic strategy, *Random* allows the algorithm to try the subset of *K*! possible orders, where *K* is the number of objects. The number of examined orders is restricted by stop conditions defined in *stopStrategiesSets* (see further).

The pseudo code of the SGDP algorithm is as follows:

Input data (see chapter 3.4.2.1): K, $G=\langle V_G, A_G\rangle$, $\mathbf{A}=[a_{ink}]_{V\times M\times K}$ – matrix of source and destination nodes via indirect nodes for each object (generating set of path segments $PS(k)$ for the k-th object, number of path segments for each object is equal M)

```
0) Save initial graph G state
1) WHILE (none of the stop conditions is fulfilled) DO
2)   Generate stripe order using stripeOrderStrategy;
3)   IF no unchecked stripe order remains -> THEN EXIT; END IF;
4a)  IF searching mode equals SameWidth THEN
5a)    WHILE (none of the stop conditions is fulfilled) DO
6a)      Generate width of stripes using
         widthOfStripeGenerationStrategy;
7a)      IF no unchecked width remains THEN
           Restore initial graph G state and go to 5a);
         END IF;
8a)      FOR each path k among K objects to find DO
9a)        FOR each segment in PS(k)
10a)           Search path for segment in G;
           END FOR;
11a)       IF path was found THEN
               save path for k object and remove used nodes and
arcs                     from graph G;
           END IF;
         END FOR;
12a)     IF for all objects paths were found THEN
           save feasible solution;
         END IF;
       END WHILE;
     END IF;
4b)  IF searching mode equals VariousWidth THEN
5b)    WHILE (none of the stop conditions is fulfilled) DO
6b)      FOR each path k among K objects DO
7b)        Generate width of stripes for k object using
             widthOfStripeGenerationStrategy
8b)        IF no unchecked width remains THEN
             Restore initial graph G state and go to 5b);
           END IF;
9b)        FOR each segment in PS(k) DO
10b)         Search path for segment in G;
           END FOR;
11b)       IF path was found THEN
             save path for k object and remove used nodes and arcs
                 from graph G
           ELSE restore initial graph G state
           END IF;
         END FOR;
12b)     IF for all objects have found paths
           save feasible solution and restore initial graph G state
         END IF;
       END WHILE;
     END IF;
   END WHILE;
```

There are two modes of path searching: *SameWidth* – all stripes must have the same width, *VariousWidth* – each stripe may have a different width.

Two different strategies for generating the width of stripes are implemented:

widthOfStripeGenerationStrategy={*Constant*, *Random*},

where *Constant* – width of the stripe is given and never changed; *Random* – width of thee stripe is randomized with a uniform distribution. Random strategy implementation generates a new width for the stripe with respect to the previous generated width, so that only greater values are allowed. Minimal width increasing is 0.5 unit (one unit = distance between two neighbouring nodes (squares)).

Additionally, we used four different stop strategies, which could be used in any combination:

stopStrategies={*MaxIterationsNumber*, *MaxFeasibleSolutionsFound*, *NextSolutionIsBetter*, *TimeLimit*}.

In the *MaxIterationNumber* strategy the algorithm ends when the maximum iteration number is reached, where a single iteration is the one searched for with a fixed order and a width of the stripes. With the *MaxFeasibleSolutionsFound* strategy the algorithm ends, when a specific number of the feasible (acceptable) solutions is found; *NextSolutionIsBetter* stop strategy ends, when the next feasible solution is no better than the previous one, plus there is a specific epsilon value. The last strategy, *TimeLimit*, stops the algorithm when the execution time reaches the specified time limit.

We can save the found paths during previous iterations or not for the objects (*PathMemory*={*true*, *false*}): if the next iteration uses the same stripe width as for the previously found path we can use it to decrease computational time of the iteration.

Let us analyse the computation complexity of the *SGDP* algorithm. Generating K subgraphs for each source-destination pairs in each path segment is an operation, which complexity is $O(MKV)$. Determining the shortest path in each subgraph has a complexity of $O(A \log V)$ using the Dijkstra's algorithm with binary heaps; since we do it MK times (M path segments for each of K objects) we have $O(MKA \log V)$. The number of possible combinations of paths determining the order is equal to the number of permutation among K elements, that is $K!$. If we check it for each possible action stripe width (let the number of the possible action stripe width for each object be equal Q) then it can be done, in the worst case, $Q^K K!$ times. Since the complexity of the *SGDP* algorithm is $O(Q^K K! MKA \log V)$. The estimation of $O(Q^K K! MKA \log V)$ of the *SGDP* complexity is only theoretical (when all stop conditions: *MaxIterationsNumber*, *MaxFeasibleSolutionsFound*, *NextSolutionIsBetter*, *TimeLimit* would have maximum values). In practice, time complexity of the *SGDP* is estimated by the function $O(WMKA \log V)$, where

W=MaxIterationsNumber and we never take into account all $Q^K K!$ possibilities of determining paths, because of using techniques to avoid checking all of them: randomization, different stop conditions, saving paths found earlier, etc. Experimental results show that in an average case the *SGDP* runs in polynomial time (see Fig. 3.24, Table 3.11: *minCT_ SGDP*, *maxCT_ SGDP*, *avgCT_ SGDP*).

Let us notice that the considered algorithm superbly fits the parallel computations by using, for example, *K* processors (each of the processors generates a subgraph and determines the shortest path in this subgraph). In such a case we accelerate computations about *K* times.

3.4.3.2. Minimal cost flow problem-based algorithms

In the paper (Tarapata, 2008e) it has been shown how to use modifications of the Busacker-Gowen minimal-cost flow algorithm (Busacker & Gowen, 1961) to solve the node-disjoint case of the problems: *DP*2.1, *DP*2.2, *DP*2.3, *DP*2.4 in some military applications.

The problem of finding an acceptable solution of *K* node-disjoint paths in the $S=\left\langle G=\left\langle V_G,A_G\right\rangle ,c\right\rangle$ network (see Fig. 3.21a) from K_s to K_t subset of nodes is based on the S^* temporary network (see Fig. 3.21b) and the maximal flow algorithm. We use the well-known conclusion from the Ford-Fulkerson theorem concerning maximal flow in network *S* from *s* (source) to *t* (target) (Wilson, 1998, pp.172): "*If the capacity value of each arc in the network S is an integer, then the capacity* c_{ij} *of the arc (i,j) describes the number of arcs linking i and j. The value of the maximal flow in such a network describes the number of all arc-disjoint paths from s to t*". Since we would like to find node-disjoint paths (instead of arc-disjoint) we must modify the *S* network to S^*. The temporary network S^* is constructed as follows:

$$S^*=\left\langle G^*,c\right\rangle \tag{3.90}$$

where $G^*=\left\langle V^*,A^*\right\rangle$, $V^*=V'\cup V''\cup\{s,t\}$, $c:V^*\times V^*\to N$ – capacity function, $c_{ij}=c(i,j)$ – capacity function value for arc (i,j). Graph G^* is constructed as follows: each node *v* of graph *G* is replaced by two nodes (see Fig. 3.21b): *v*' (belonging to *V*' set) and *v*" (belonging to *V*" set), next we link *v*' and *v*" by an arc and set capacity of this arc equal to 1. All of the arcs, which end in the *v* node in graph *G* will end in *v*' node in graph G^*, all of the arcs which start in the *v* node in *G* will start from *v*" in G^*. Each of these arcs in G^* will have the same capacity as in *G*. Moreover, two nodes (*s* and *t*) are added to the set of nodes in G^*: we link node *s* with each of the nodes belonging to the K_s set and each of the nodes belonging to K_t with node *t*. Each of these arcs has the capacity value set to 1.

The problem of finding the *K* node-disjoint paths in network S^* is based on the maximal flow problem definition. Flow *f* in network S^* is a function, which set for each arc (i,j) in S^* such a nonnegative value f_{ij} that:

1. for each arc (i,j) in S^* the following formula is fulfilled:

$$0 \le f_{ij} \le c_{ij} \tag{3.91}$$

2. for source node s in S^* the following formula is fulfilled:

$$\sum_{i \in V^*} f_{si} - \sum_{i \in V^*} f_{is} = FV \tag{3.92}$$

   Value FV is called the flow value.

3. for target node t in S^* the following formula is fulfilled:

$$\sum_{i \in V^*} f_{ti} - \sum_{i \in V^*} f_{it} = -FV \tag{3.93}$$

4. for each node $j \in V^* / \{s,t\}$:

$$\sum_{i \in V^*} f_{ji} - \sum_{i \in V^*} f_{ij} = 0 \tag{3.94}$$

Maximal flow problem is defined as follows: for a given S^*, s, t to find $[f_{ij}]^*$ which satisfy (3.91)-(3.94) and

$$FV([f_{ij}]^*) = \max_{[f_{ij}] \in SF(s,t)} FV([f_{ij}]) \tag{3.95}$$

where $SF(s,t)$ – set of all possible flows in S^* from s to t.

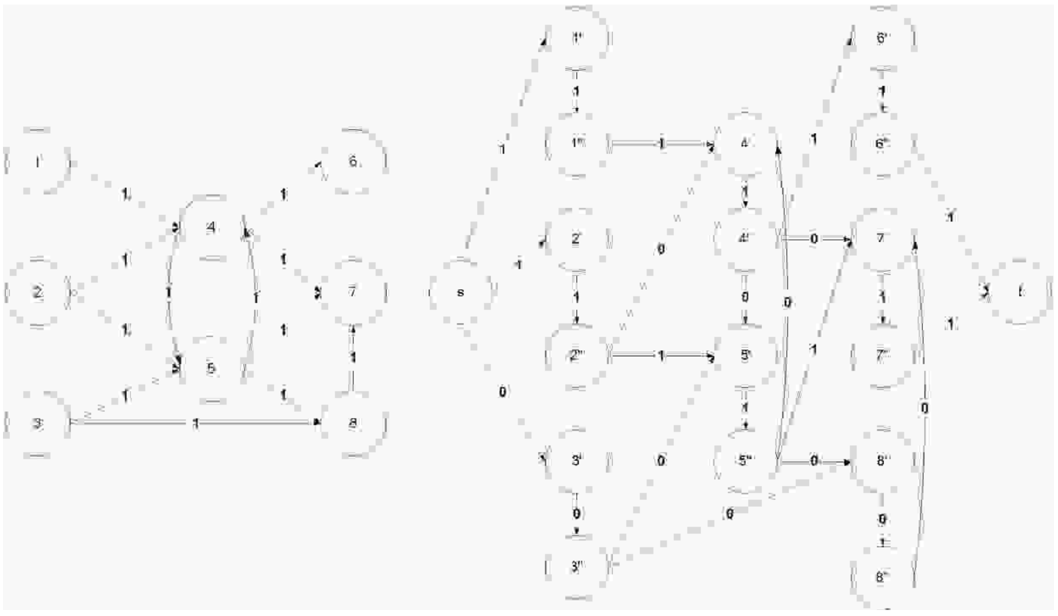


Fig. 3.21. (a) Network S with values of arc capacity c_{ij}; (b) Network S^* related to S with flow values $[f_{ij}]^*$ after realization of two iterations of maximal flow algorithm (FV=2), K_s={1,2, 3}, K_t={6,7}

If, after solving the problem, $F([f_{ij}]^*)<K$ then in S^* (and in consequence in S) K node-disjoint paths from K_s to K_t does not exist. Otherwise, K node-disjoint paths from K_s to K_t exist and we can read them after the last step of the maximal flow

algorithm as follows: we K times start from s and choice arcs (i,j) with $f_{ij}=1$ till we achieve t. These alternate sequences of nodes and arcs indicate the k-th disjoint path from s to t. For example, in Fig. 3.21b we have $FV=K=2$ disjoint paths (as the sequence of nodes) from $K_s=\{1,2,3\}$ to $K_t=\{6,7\}$: (1st) 1'-1"-4'-4"-6'-6"; (2nd) 2'-2"-5'-5"-7'-7".

To find the K node-disjoint *shortest* paths (with minimal total cost of all K paths) we modify the S^* network as follows:

$$S^{**} = \left\langle G^*, \{q, c\} \right\rangle \tag{3.96}$$

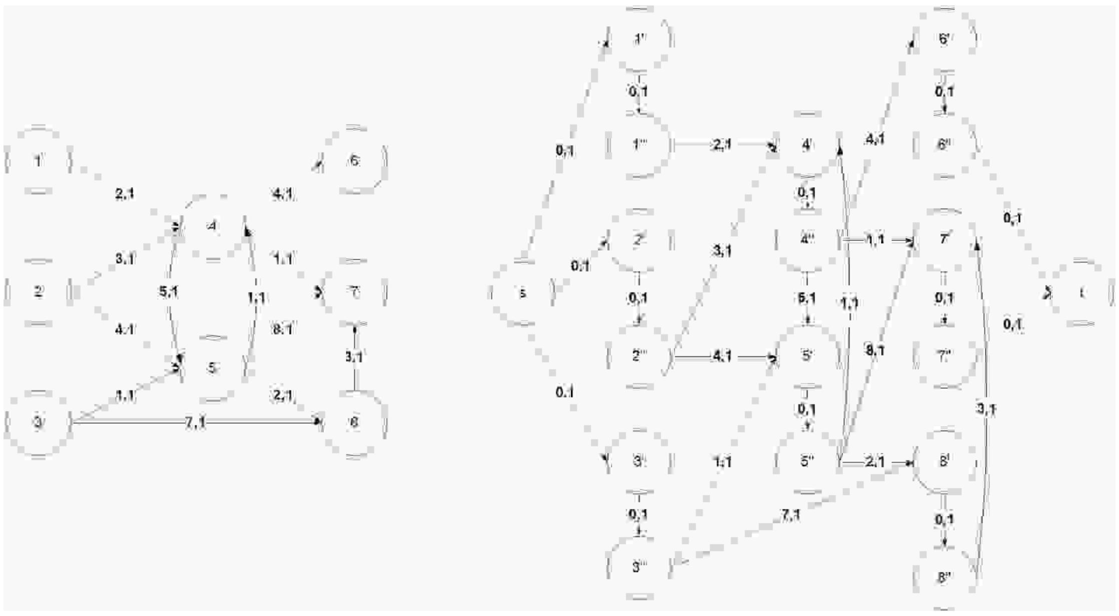
where $q : V^* \times V^* \to R^+$ – time cost function, $q_{ij}=q(i,j)$ – value of the function for arc (i,j).

Formulation of the K node-disjoint *shortest* paths problem in the S^* network from s to t defined as minimal cost flow problem with demanded flow equal K is as follows:

$$\sum_{(i,j)\in A^*} q_{ij} f_{ij} \to \min \tag{3.97}$$

with constraints: (3.91), (3.92) where we replace $FV=K$, (3.93) where we replace $FV=K$, (3.94).

Method for solving this problem is based on the Busacker-Gowen algorithm (Busacker & Gowen, 1961) with a complexity of $O(V^4)$, where V is the number of nodes in G^* and presented in details in (Tarapata, 2008e).



(a) (b)

Fig. 3.22. (a) Network S with values of q_{ij} and c_{ij}; (b) Network S^{**} related to S prepared for finding $FV=K=2$ node-disjoint shortest paths, $K_s=\{1, 2, 3\}$, $K_t=\{6, 7\}$

Let us note that the above-presented method does not guarantee solving the problem when K_s and K_t are vectors (and we must find the K node-disjoint paths between K pairs of specified nodes). This method only allows finding K disjoint paths between any of the nodes belonging to K_s and K_t , so it can be used for solving node-disjoint case problems: *DP*2.1, *DP*2.2, *DP*2.3, *DP*2.4.

In the paper (Tarapata & Wroclawski, 2011d) it has been shown how to use modifications of the Edmonds-Karp (Edmonds & Karp, 1972) algorithm to solve the *NDRP-Sum* and *NDRP-Max* problems (modifications of the *DP*2.5 problem). A specific method for constructing the temporary network being used in the modified Edmonds-Karp algorithm in order to find the K node-disoint paths visiting specified nodes has been proposed.

3.4.4. Experimental Analysis of the Algorithms

We have conducted computations for real terrain areas used in *Zlocien* system with different number of nodes: 5 000, 7 540 (Fig. 3.23a), 10 000, 20 500, 25 000 (Fig. 3.23b) and 35 000. We have used random pairs of source-destination nodes (single segments only (M=1)) for K objects ($K \in \{2, 3, 4, 5, 6\}$). We have performed research for almost every possible combination of the *SGDP* algorithm parameters defined in chapter 3.4.3.1 for the *NDRP-Sum* problem.

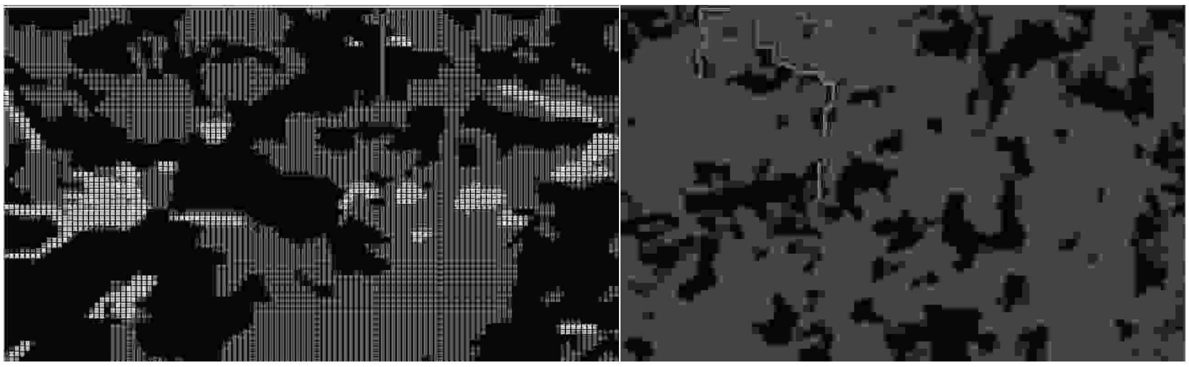


Fig. 3.23. Typical mesh graphs representing a fragment of the terrain. Colour represents cost of nodes: the light colour of the nodes (square) describes open (well passable) terrain, the dark colour describes obstacles (forests, lakes, rivers, buildings), the lighter the colour the smaller the cost value. (a) Graph with 7 540=65×116 nodes representing terrain near Drawsko (Poland). (b) Graph with 25 000=125×200 nodes representing terrain near Radom (Poland) with two node-disjoint paths found by the *SGDP* algorithm (lighter colour)

The following *stopStrategiesSets* have been used: {{*TimeLimit*, *MaxIterationNumber*}, {*TimeLimit*, *MaxIterationNumber*, *NextSolutionIsBetter*}, {*TimeLimit*, *MaxIterationNumberStrategy*, *NFeasibleSolutionsFound*}} with the following values: *MaxIterationNumber*=10, *NFeasibleSolutionsFound*=4, in *NextSolutionIsBetter* we set the minimum decrement of cost to 5.0, in *TimeLimit* strategy we have restricted the execution time of each iteration to 5 000ms.

All computations have been done using a computer with Intel Core 2 Duo 2.2 GHz processor and 3GB RAM.

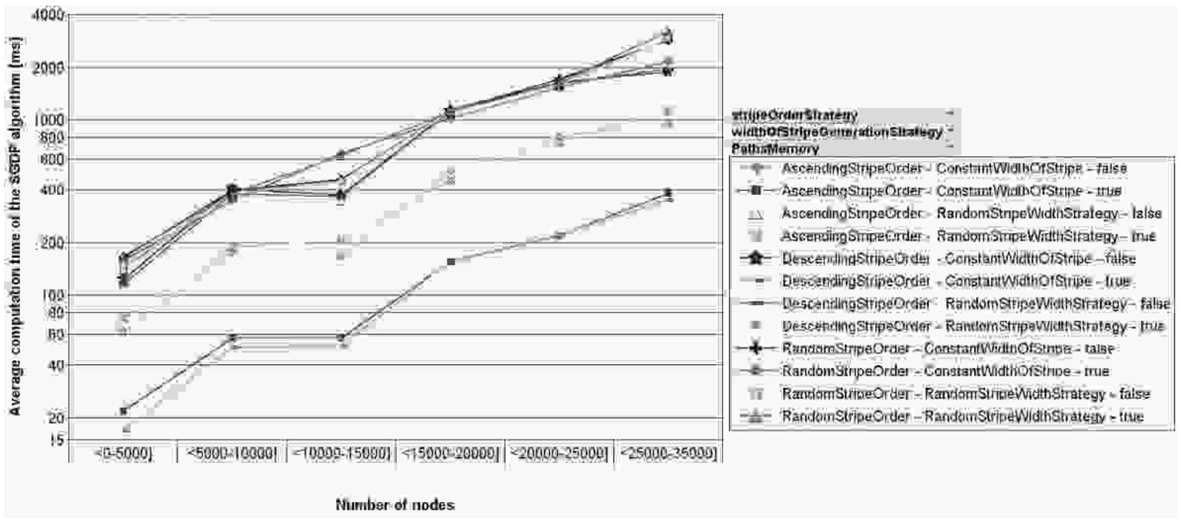


Fig. 3.24. Average computation time (milliseconds, logarithmic scale) of the *SGDP* algorithm in relation to *stripeOrderStrategy*, *widthOfStripeGenerationStrategy* and *PathsMemory*

In Fig. 3.24 we present the average computation time of the *SGDP* algorithm in relation to *stripeOrderStrategy*, *widthOfStripeGenerationStrategy* and *PathsMemory* for different number of nodes. From Fig. 3.24 results that in an average case the complexity of the *SGDP* algorithm is time-polynomial and it is much better than the theoretical estimation given in chapter 3.4.3.1. It results from using some techniques (described in chapter 3.4.3.1) to decrease this complexity, such as: randomization, different stop conditions, saving paths found earlier, etc. It is easy to notice that we have obtained the shortest computation times for *stripeOrderStrategy*∈{*Ascending*, *Descending*} and *widthOfStripeGenerationStrategy* =*Constant*. Moreover, we can notice that for each pair *stripeOrderStrategy*-*widthOfStripeGenerationStrategy* computation time for *PathsMemory*=*true* is significantly shorter than for *PathsMemory*=*false* (from about 3 to 10 times). It results from the fact that we have saved paths found during previous iterations for objects and if the next iteration uses the same stripe width as for the previously found path, we will use these paths to decrease the computational time of the iteration.

Analysis of results in Fig. 3.25 supplements the results presented above for different values of *stopStrategy*. We obtained the shortest computation time for the set of stop strategies {*TimeLimit*, *MaxIterationNumber*, *NextSolutionIsBetter*}.

In Fig. 3.26 we presented the average computation time (milliseconds, logarithmic scale) of the *SGDP* algorithm in relation to the number of graph nodes (*V*) and number of objects (*K*).

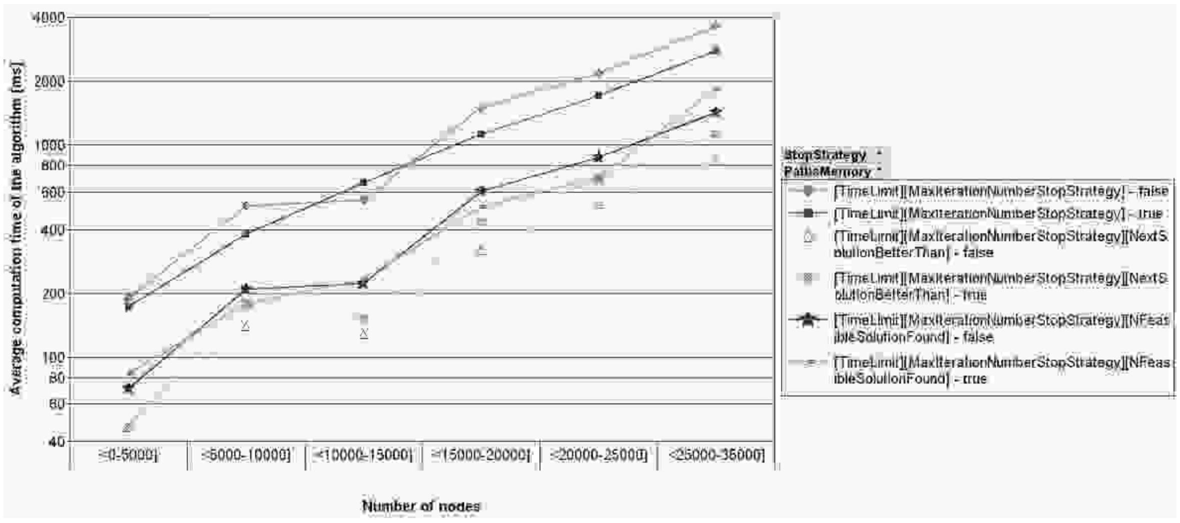


Fig. 3.25. Average computation time (milliseconds, logarithmic scale) of the *SGDP* algorithm in relation to *stopStrategy* and *PathsMemory*

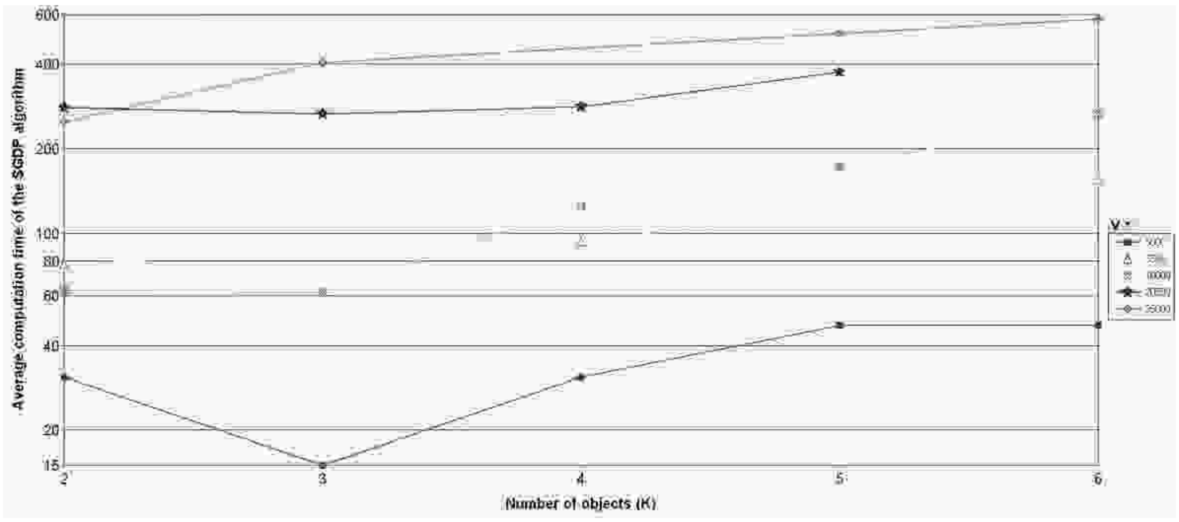


Fig. 3.26. Average computation time (milliseconds, logarithmic scale) of the *SGDP* algorithm in relation to the number of graphs nodes (*V*) and number of objects (*K*)

In Fig. 3.27 we presented the average accuracy coefficient *avgAC* of the *SGDP* algorithm (*AC*=value of the objective function obtained from the *SGDP* algorithm/optimal value of the objective function) in relation to:

stripeOrderStrategy, *widthOfStripeGenerationStrategy* and *PathsMemory*

(an accurate (optimal) solution of the problem (3.78)-(3.84) obtained using the the GAMS/CPLEX 12.2 solver). The value of *avgAC* fluctuates from ~1.02 to ~1.6. It means that the value of the objective function (sum of the cost of the *K* paths) obtained from the *SGDP* algorithm was worse from ~2% to ~60% in relation to the optimal solution (see also Table 3.11). The *SGDP* algorithm gives the best values of the *avgAC* for *PathsMemory=true*, *stripeOrderStrategy=Random*, and *widthOfStripeGenerationStrategy=Constant*.

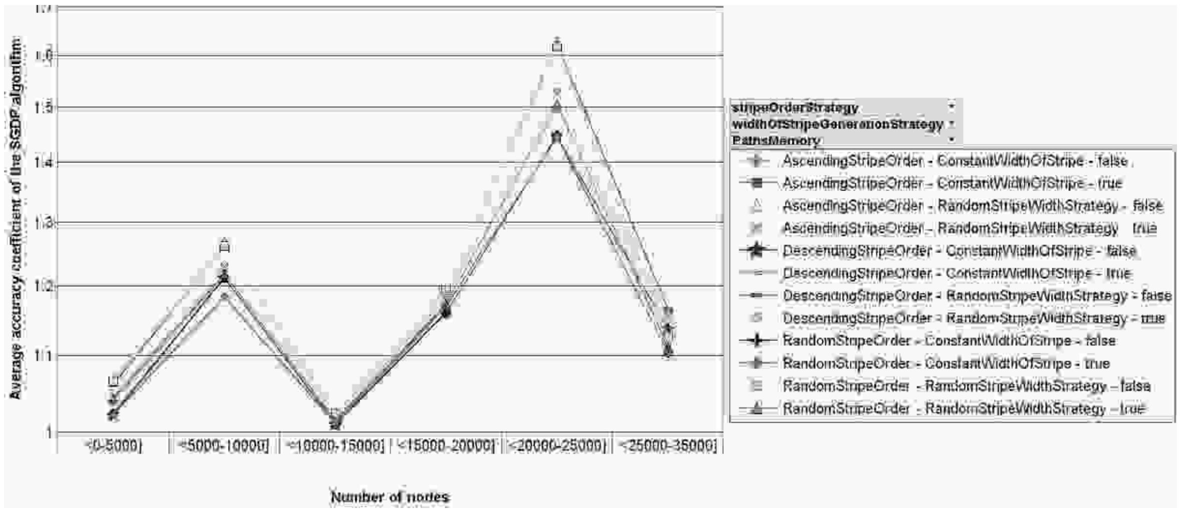


Fig. 3.27. Average accuracy coefficient (*avgAC*) of the *SGDP* algorithm in relation to *stripeOrderStrategy*, *widthOfStripeGenerationStrategy* and *PathsMemory* (accurate solution obtained using the GAMS/CPLEX 12.2 solver)

Table 3.11. Comparison of the computation time and accuracy of the *SGDP* algorithm with characteristics of the optimal solution obtained by solving problem (3.78)-(3.84) using the GAMS/CPLEX 12.2 solver for *K*=2

V	*5 000*	*7 540*	*10 000*	*20 500*	*25 000*	*35 000*
minOF_SGDP	162	1 247	355	203	2 729	23 775
maxOF_SGDP	191	1 464	362	208	3 334	25 488
avgOF_SGDP	167	1 333	356	204	2 950	24 364
OOF	158	956	353	190	1 955	21 038
minAC	2.2%	30.4%	0.7%	6.7%	39.6%	13.0%
maxAC	20.4%	53.1%	2.5%	8.9%	70.5%	21.2%
avgAC	5.5%	39.4%	1.0%	7.3%	50.9%	15.8%
minCT_SGDP	10	31	15	140	109	250
maxCT_SGDP	156	375	282	1 375	1 250	3 079
avgCT_SGDP	98.2	224.4	180.2	852.4	789.2	1 786.7
CT_CPLEX	5 200	8 910	10 690	73 140	26 030	114 410
minCTAC	33	24	38	53	21	37
maxCTAC	520	287	713	522	239	458
avgCTAC	52.9	39.7	59.3	85.8	33.0	64.0

In Table 3.11 we presented a comparison of the computation time and accuracy of the *SGDP* algorithm with characteristics of the optimal solution obtained by the solving problem (3.78)-(3.84) using the GAMS/CPLEX 12.2 solver for *K*=2. We used the following notations: *OOF* – optimal value of objective function (3.78); *minOF_SGDP*, *maxOF_SGDP*, *avgOF_SGDP* – minimal, maximal and average values of the objective function, respectively, obtained from the *SGDP* algorithm; *AC* – percentage approximation coefficient of the *SGDP* algorithm=(value of the objective function from the *SGDP* algorithm/optimal value of the objective function) in percents-100%; *minAC*, *maxAC*, *avgAC* – minimal, maximal and average values of *AC*; *minCT_SGDP*, *maxCT_SGDP*, *avgCT_SGDP* – minimal, maximal and average values of the computation time (in msec) using the *SGDP* algorithm; *CT_CPLEX* – computation time (in msec) for finding the optimal solution using the GAMS/CPLEX 12.2 solver; *minCTAC*, *maxCTAC*, *avgCTAC* – computation time acceleration coefficient *CTAC* values (respectively: minimal, maximal, average), *CTAC*=computation time using the GAMS/CPLEX solver/computation time using the *SGDP* algorithm. Values of *minAC*, *maxAC* and *avgAC* indicate that the value of the objective function (sum of the cost of the *K* paths) obtained from the *SGDP* algorithm was worse from ~1% to ~50% (average) in relation to the optimal solution, but the computation time for the *SGDP* algorithm was shorter from ~30 to ~85 times in relation to the GAMS/CPLEX solver (in selected cases even >700 times faster, *maxCTAC*(10000)=713). It is possible to increase the accuracy of the algorithm by changing its input parameters (in parenthesis we give the values, which have been used during experiments): *MaxIterationNumber* (10), *NFeasibleSolutionsFound* (4), *NextSolutionIsBetter* (5.0), *TimeLimit* (5000ms). Additional experiments have shown that by increasing, for example *MaxIterationNumber* or *TimeLimit*, we can increase the accuracy of the algorithm, but at the cost of time. Parameter values, which have been used during experiments described in this chapter made some compromise between accuracy and time-complexity of the *SGDP* algorithm.

3.5. Summary

As it has been written in chapter 3.1, all presented methods have applications in many transportation problems, especially ones related to paths planning. The approach presented in chapter 3.2 is dedicated especially for multiresolution path planning in grid graph-based route planning when the grid represents, for example a terrain environment as a regular grid of terrain squares. It can be shown that a multiresolution approach for path planning represented by finding shortest paths in recurrently defined G^* can also be used for multistage path planning: we can first find a "rough" path in a "rough" terrain represented by G^* (for example in Fig. 3.11) and then we can find an accurate path in a more detailed environment. However, the *DSP* algorithm gives a good result not only for the all-pairs shortest paths problem (Table 3.4). Since the most complex steps of the algorithm (steps 1-3, "bottleneck") are done only one time (the b-graph is built only one time – initial preprocessing), then if we compute a one-pair shortest path many times it allows us to shorten the time of the "bottleneck". It is also possible to set a compromise between space and time complexity of the *DSP* algorithm.

However, any algorithm solving the multiobjective shortest path problem is, at least, exponential in the worst case analysis but we can use specific, effective approaches for the special *MOSP* problems. In chapter 3.3 we focused on analysis of complexity of selected *MOSP* problems and showed how we can use modifications and advantage of fast implementations of the Dijkstra's algorithm in order to effectively and optimally solve them. Experimental results of the computational time for the presented approach (especially the modified Dijkstra's algorithm) in chapter 3.3.5 confirm their good effectiveness for solving selected *MOSP* problems. Models and methods described in the chapter were selected from numerous approaches. Such problems as: determining disjoint paths (Li *et al.*, 1992, Schrijver & Seymour, 1992; Tarapata, 1999a; 2000e), stochastic network dependencies (Sigal *et al.*, 1980; Korzan, 1982; 1983a; 1983b; Loui, 1983), time-dependencies in the network (Bernstein & Kelly, 1997; Cai *et al.*, 1997; Djidjev *et al.*, 1995; Sherali *et al.*, 1998) in multicriteria context were only indicated here.

Algorithms presented in chapter 3.4 (*SGDP* and modifications of minimal cost flow algorithms) for solving the node-disjoint shortest *K* paths problem in mesh graphs can be used for transportation, e.g. maneuver planning of military detachments (Tarapata, 2009a). For one of them (*SGDP*) it has been shown that it is fast (in comparison with the GAMS/CPLEX solver) and gives a satisfying solution to the problem (experimental average approximation coefficient of the algorithm is equal from 1% to 50%). Since the algorithm is approximated it seems to be essential to provide necessary and sufficient conditions for obtaining optimal solutions and estimate the theoretical approximation coefficient. Moreover, it seems to be essential to examine sensitivity of the algorithm changing number of indirect nodes in paths for each object and values of the parameters: *MaxFeasibleSolutionsFound*, *NextSolutionIsBetter*, *NFeasibleSolutionsFound*. It is possible to extend the considered problem using more criteria (e.g. minimization of maximal path cost for any object) and obtaining the multicriteria disjoint shortest paths problem.

Majority of the presented methods have been used in practice. Many very interesting models for paths planning (alternative paths, simplest path, time-dependent paths) have not been presented here due to limitation reasons and can be found in other papers of the author (Tarapata, 2004b; 2006c; Tarapata *et al.*, 2009b; 2009d; Tarapata & Mierzejewski, 2010f). However, some of these applications are presented in chapter 6.

Appendix 3.A.1. Proof of Theorem 3.1

The proof consists of three parts: in the first one we consider a case when $G^*=G$, in the second one we prove that $L(d^{\min}(s,t)) \geq L^{*\min}(d^{*\min}(x_s^*, x_t^*))$ and the third one contains proof that $L(d^{\min}(s,t)) \leq L^{*\max}(d^{*\max}(x_s^*, x_t^*)) + L'(s, W(x_s^*, x_1^*))$.

Part 1

If $V^*=V$ then $G^*=G$ and each $x^*=x$. Moreover, for each $x^*, y^* \in V_G^*$ occurs: $W(x^*, y^*) = \{x\}$, hence for each $z^*, y^* \in V_G^*$ the following formulas are true:

$$c_{z^*}^{*\min}(x^*, y^*) = \min_{d(\cdot,\cdot)\in D^{\min}(x,x)} L(d(\cdot,\cdot)) + \min_{d(\cdot,\cdot)\in D^{\min}(x,y)} L(d(\cdot,\cdot)) = 0 + c(x,y),$$

$$c_{z^*}^{*\max}(x^*, y^*) = \max_{d(\cdot,\cdot)\in D^{\min}(x,x)} L(d(\cdot,\cdot)) + \max_{d(\cdot,\cdot)\in D^{\min}(x,y)} L(d(\cdot,\cdot)) = 0 + c(x,y)$$

because of $D^{\min}(x,x) = \{x\}$ and $D^{\min}(x,y) = \{d^{\min}(x,y)\} = \{(x,y)\}$ represents arc from x to y, hence $L(d^{\min}(x,y)) = c(x,y)$. In such case $L(d^{\min}(s,t)) = L^{*\min}(d^{*\min}(x_s^*, x_t^*))$ = $L^{*\max}(d^{*\max}(x_s^*, x_t^*))$ and formula (3.7) is fulfilled.

Part 2

Now, Let G^* be a graph with $n<V$ (n – count of nodes in G^*) nodes. Let us take into account the shortest path $d^{\min}(s,t) = (x_0 = s, x_1, x_2, ..., x_{l(d(s,t))} = t)$ in G from s to t. Each path $d(s,t)$ in G "generates" path $d^*(x_s^*, x_t^*)$ in G^* such, that $\underset{i\in\{0,...,l(d(s,t))\}}{\forall} \ \underset{j\in\{0,...,l^*(d^*(x_s^*,x_t^*))\}}{\exists} x_i \in x_j^*$ and $d^*(x_s^*, x_t^*) = (x_0^* = x_s^*, x_1^*, x_2^*, ..., x_{l^*(d^*(x_s^*,x_t^*))}^* = x_t^*)$. Let us determine for each x_i^*, $i = 0,...,l^*(d^*(x_s^*, x_t^*))$ set $T(x_i^*) = \{x \in d(s,t) : x \in x_i^*\}$. For example, for graph G^* in Fig. 3.2 we have:

$d^{\min}(s=9, t=8) = (9,10,12,3,5,7,8)$, $d^*(x_s^*, x_t^*) = (x_0^* = E, x_1^* = A, x_2^* = B)$ and $T(x_0^*) = \{9,10,12\}$, $T(x_1^*) = \{3\}$, $T(x_2^*) = \{5,7,8\}$. Let us consider any two neighbouring b-nodes $x_i^*, x_{i+1}^* \in d^*(x_s^*, x_t^*)$, $i>0$. Let us order nodes belonging to $T(x_i^*)$ and $T(x_{i+1}^*)$ topologically ($|T(□)|$ describes cardinality of the set $T(□)$). We obtain for $T(x_i^*)$ and $T(x_{i+1}^*)$ topologically ordered sequences of nodes: $x_{i,1}^*, x_{i,2}^*, ..., x_{i,|T(x_i^*)|}^*$ and $x_{i+1,1}^*, x_{i+1,2}^*, ..., x_{i+1,|T(x_{i+1}^*)|}^*$. Let us take the first $x_{i,1}^*$ and the last $x_{i,|T(x_i^*)|}^*$ nodes from $T(x_i^*)$ and the first node $x_{i+1,1}^*$ from $T(x_{i+1}^*)$. It is easy to observe that:

$$x_{i,1}^* \in W(x_i^*, x_{i-1}^*),\quad x_{i,|T(x_i^*)|}^* \in W(x_i^*, x_{i+1}^*)\ ,\quad x_{i+1,1}^* \in W(x_{i+1}^*, x_i^*) \qquad (3.A.1)$$

For example (see Fig. 3.2), for $T(x_{i=1}^*) \Rightarrow x_{1,1}^* = 3$, for $T(x_{i+1=2}^*) \Rightarrow x_{2,1}^* = 5$, $x_{2,2}^* = 7$, $x_{2,3}^* = 8$ and next $x_{1,1}^* = 3 \in W(x_1^*, x_0^*) = \{1,3\}$, $x_{2,1}^* = 5 \in W(x_2^*, x_1^*) = \{5,6\}$, $x_{1,|T(x_i^*)|}^* = x_{1,1}^* = 3 \in W(x_1^*, x_2^*) = \{3,4\}$.

We will show that $L(d^{\min}(x^*_{i,1},x^*_{i+1,1}))\geq L^{*\min}(d^{*\min}(x^*_i,x^*_{i+1}))$.

Let us observe that $d^{\min}(x^*_{i,1},x^*_{i+1,1})=d^{\min}(x^*_{i,1},x^*_{i,|T(x^*_i)|})\,||\,d^{\min}(x^*_{i,|T(x^*_i)|},x^*_{i+1,1})$ where symbol "| |" denotes the concatenation of two paths, hence

$$L(d^{\min}(x^*_{i,1},x^*_{i+1,1}))=L(d^{\min}(x^*_{i,1},x^*_{i,|T(x^*_i)|}))+L(d^{\min}(x^*_{i,|T(x^*_i)|},x^*_{i+1,1})) \tag{3.A.2}$$

Next, let us note that from (3.5) results: $L^{*\min}(d^{*\min}(x^*_i,x^*_{i+1}))=c^{\min}_{x^*_{i-1}}(x^*_i,x^*_{i+1})$ for i>0 because x^*_i, x^*_{i+1} are adjacent in G^*, and the length of the path from x^*_i to x^*_{i+1} equals the length of the arc between these nodes, that is $c^{\min}_{p(x^*_i)}(x^*_i,x^*_{i+1})$ and $p(x^*_i)$ equals like in (3.5). From (3.2) we have:

$$\begin{aligned}c^{\min}_{x^*_{i-1}}(x^*_i,x^*_{i+1})=&\min_{d(\cdot,\cdot)\in D^{\min}\left(W(x^*_i,x^*_{i-1}),W(x^*_i,x^*_{i+1})\right)}L(d(\cdot,\cdot))+\\&+\min_{d(\cdot,\cdot)\in D^{\min}\left(W(x^*_i,x^*_{i+1}),W(x^*_{i+1},x^*_i)\right)}L(d(\cdot,\cdot))\end{aligned} \tag{3.A.3}$$

Let us consider the first elements of sums in (3.A.2) and (3.A.3). It is easy to observe that

$$\min_{d(\cdot,\cdot)\in D^{\min}\left(W(x^*_i,x^*_{i-1}),W(x^*_i,x^*_{i+1})\right)}L(d(\cdot,\cdot))\leq L(d^{\min}(x^*_{i,1},x^*_{i,|T(x^*_i)|})) \tag{3.A.4}$$

because of (3.A.1) we obtain: $d^{\min}(x^*_{i,1},x^*_{i,|T(x^*_i)|})\in D^{\min}\left(W(x^*_i,x^*_{i-1}),W(x^*_i,x^*_{i+1})\right)$ and the inequality (3.A.4) is clear. Let us consider the second elements of sums in (3.A.2) and (3.A.3). By analogy we obtain:

$$\min_{d(\cdot,\cdot)\in D^{\min}\left(W(x^*_i,x^*_{i+1}),W(x^*_{i+1},x^*_i)\right)}L(d(\cdot,\cdot))\leq L(d^{\min}(x^*_{i,|T(x^*_i)|},x^*_{i+1,1})) \tag{3.A.5}$$

Taking into account (3.A.1) we have:

$$d^{\min}(x^*_{i,|T(x^*_i)|},x^*_{i+1,1})\in D^{\min}\left(W(x^*_i,x^*_{i+1}),W(x^*_{i+1},x^*_i)\right)$$

and inequality (3.A.5) is clear. From (3.A.4) and (3.A.5) results

$$L(d^{\min}(x^*_{i,1},x^*_{i+1,1}))\geq L^{*\min}(d^{*\min}(x^*_i,x^*_{i+1}))=c^{\min}_{x^*_{i-1}}(x^*_i,x^*_{i+1})$$

for each i>0. For i=0 we have to examine condition: $L(d^{\min}(x^*_{0,1},x^*_{1,1}))\geq L^{*\min}(d^{*\min}(x^*_0,x^*_1))$. We can then again write:

$$d^{\min}(x^*_{0,1},x^*_{1,1})=d^{\min}(x^*_{0,1},x^*_{0,|T(x^*_0)|})\,||\,d^{\min}(x^*_{0,|T(x^*_0)|},x^*_{1,1})$$

$$L(d^{\min}(x^*_{0,1},x^*_{1,1}))=L(d^{\min}(x^*_{0,1},x^*_{0,|T(x^*_0)|}))+L(d^{\min}(x^*_{0,|T(x^*_0)|},x^*_{1,1}))$$

Next, from (3.5) results that

$$L^{*\min}(d^{*\min}(x_0^*,x_1^*)) = c_{x_1^*}^{\min}(x_0^*,x_1^*) = \min_{d(\cdot,\cdot)\in D^{\min}\left(W(x_0^*,x_1^*),W(x_1^*,x_0^*)\right)} L(d(\cdot,\cdot))$$,

hence $L(d^{\min}(x^*_{0,|T(x_0^*)|},x^*_{1,1})) \geq \min_{d(\cdot,\cdot)\in D^{\min}\left(W(x_0^*,x_1^*),W(x_1^*,x_0^*)\right)} L(d(\cdot,\cdot))$.

We have shown that condition $L(d^{\min}(x^*_{i,1},x^*_{i+1,1})) \geq L^{*\min}(d^{*\min}(x_i^*,x_{i+1}^*))$ is fulfilled for each $i=0,...,l^*(d^*(x_s^*,x_t^*))-1$, hence $L(d^{\min}(s,t)) \geq L^{*\min}(d^{*\min}(x_s^*,x_t^*))$ from (3.7) is fulfilled.

Part 3

To prove that $L(d^{\min}(s,t)) \leq L^{*\max}(d^{*\max}(x_s^*,x_t^*))$ we will first show that $L(d^{\min}(x^*_{i,1},x^*_{i+1,1})) \leq L^{*\max}(d^{*\max}(x_i^*,x_{i+1}^*))$, $i=1,...,l^*(d^*(x_s^*,x_t^*))-1$, by analogy to part 2. From (3.A.2), (3.5) and (3.6) results: $L^{*\max}(d^{*\max}(x_i^*,x_{i+1}^*)) = c_{x_{i-1}^*}^{\max}(x_i^*,x_{i+1}^*)$ and from (3.3)

$$c_{x_{i-1}^*}^{\max}(x_i^*,x_{i+1}^*) = \max_{d(\cdot,\cdot)\in D^{\min}\left(W(x_i^*,x_{i-1}^*),W(x_i^*,x_{i+1}^*)\right)} L(d(\cdot,\cdot)) + \\ + \max_{d(\cdot,\cdot)\in D^{\min}\left(W(x_i^*,x_{i+1}^*),W(x_{i+1}^*,x_i^*)\right)} L(d(\cdot,\cdot)) \tag{3.A.6}$$

It is easy to notice, by analogy to (3.A.4) and (3.A.5), that the first element of the sum from (3.A.6) is greater than the first element of the sum from (3.A.2) and the second element of the sum from (3.A.6) is greater than the second element of the sum from (3.A.2), hence $L(d^{\min}(x^*_{i,1},x^*_{i+1,1})) \leq L^{*\max}(d^{*\max}(x_i^*,x_{i+1}^*))$, $i=1,...,l^*(d^*(x_s^*,x_t^*))-1$. For *i*=0 we have to examine condition:

$$L(d^{\min}(x^*_{0,1},x^*_{1,1})) \leq L^{*\max}(d^{*\max}(x_0^*,x_1^*)) + L'(s,W(x_0^*,x_1^*))$$

But $d^{\min}(x^*_{0,1},x^*_{1,1}) = d^{\min}(x^*_{0,1},x^*_{0,|T(x_0^*)|}) \,||\, d^{\min}(x^*_{0,|T(x_0^*)|},x^*_{1,1})$ and

$$L(d^{\min}(x^*_{0,1},x^*_{1,1})) = L(d^{\min}(x^*_{0,1},x^*_{0,|T(x_0^*)|})) + L(d^{\min}(x^*_{0,|T(x_0^*)|},x^*_{1,1}))$$

Next, from (3.3) and (3.5) results that $L^{*\max}(d^{*\max}(x_0^*,x_1^*)) = c_{x_1^*}^{\max}(x_0^*,x_1^*) = \max_{d(\cdot,\cdot)\in D^{\min}\left(W(x_0^*,x_1^*),W(x_1^*,x_0^*)\right)} L(d(\cdot,\cdot))$. Since $L'(s,W(x_0^*,x_1^*))$ is the length of the longest of the shortest paths from $s=x^*_{0,1}$ to any node from $W(x_0^*,x_1^*)$ hence $L'(s,W(x_0^*,x_1^*)) \geq L(d^{\min}(x^*_{0,1},x^*_{0,|T(x_0^*)|}))$. By analogy,

$$\max_{d(\cdot,\cdot)\in D^{\min}\left(W(x_0^*,x_1^*),W(x_1^*,x_0^*)\right)} L(d(\cdot,\cdot)) \geq L(d^{\min}(x^*_{0,|T(x_0^*)|},x^*_{1,1}))$$

Thus, we have shown that condition:

$$L(d^{\min}(s,t)) \leq L^{*\max}(d^{*\max}(x_s^*,x_t^*)) + L'(s,W(x_s^*,x_1^*))$$

is fulfilled.

Q.E.D. ♦

4. Models and Algorithms for Movement Synchronization

4.1. Introduction

Scheduling movement of objects is an essential element of numerous systems: for routing in computer networks (Cidon *et al.*, 1997; 1999; Kerbache & Smith, 2000; Silva & Craveirinha, 2004; Tarapata, 2006a), for movement planning of mobile robots (Buchli, 2006; Jing, 2008; Ozaki *et al.*, 1993), for tasks processed inside distributed or parallel computing systems (Leung, 2004; Tarapata, 1999a; 2000e), for redeployment of military detachments (Logan, 1997a; Rajput & Karr, 1994; Tarapata, 1999b; 2000b; 2000f; 2001; 2003a; 2004b; 2004c; 2005a; 2005b; 2007a; 2007e; 2008a; 2008b; 2008c; 2008d; 2009a; 2010b, 2011b), in crowd planning and simulation (Klupfel *et al.*, 2005; Najgebauer *et al.*, 2009) or in computer games (Van der Akker *et al.*, 2010), etc. The movement synchronization scheduling (*MSS*) problem deals with planning of movement for many objects to synchronize their movement. This problem most often consists of two subproblems: (*MSS1*) paths planning for many objects; (*MSS2*) movement organization by determining synchronization checkpoints and times on the paths. The *MSS1* problem has been analysed in detail in chapter 3. The *MSS2*, e.g. in military applications, results from the fact that objects (tanks, trucks, aircrafts, units, convoys) are moved according to a group pattern. From the point of view of mission realization, preservation of group pattern during military actions is very important: each object being moved in a group (e.g. during attack, during redeployment) must keep specific distances between each other inside the group (Logan, 1997a; Tarapata, 2011a) or must achieve specific checkpoints in given times (Tarapata, 2009a). Taking into account military applications (e.g. battlefield simulation systems, military logistics systems), movement synchronization scheduling has an influence on accuracy, adequacy, effectiveness and other characteristics of such systems. Afterwards, the problem is to model and optimize such movements of detachments to achieve intended goals of commands (such as: achievement of destinations on restricted time, avoiding losses during redeployment etc.). A special type of system with this requirement is the *Allied Deployment and Movement System* (*ADAMS*) (Heal & Garnett, 2001), which has been developed in support of multinational force movement planning in NATO. The *ADAMS* provides the users with the tools to plan and manage deployment operations. The other example of using such requirements are modules for movement planning and simulation of military objects (units) in combat simulators (Ceranowicz, 1994; Campbell *et al.*, 1995, Logan, 1997a; Henninger *et al.*, 2000; Longtin & Megherbi, 1995; Najgebauer *et al.*, 2007b; Rajput & Karr, 1994, Reece *et al.*, 2000; Tarapata, 2000c; 2010b).

This chapter is organized as follows. In chapter 4.2 selected scheduling models and algorithms for synchronous movement are described. Some properties of these algorithms are proved. Experimental analysis of the algorithms has been given. In chapter 4.3 two-criteria movement synchronization scheduling problem has been defined. Method for solving the problem has been described. Presented models and algorithms are based on the papers (Tarapata 2001; 2005b; 2007a; 2008a; 2008d; 2009a; 2010b).

4.2. Movement Synchronization Scheduling (*MSS*)

4.2.1. Scheduling Models of Synchronous Movement

4.2.1.1. Notations and definitions

Let us assume that we have a directed graph G that defines the structure of the terrain (divided into squares, hexagons - see chapter 2), $G=\langle V_G, A_G\rangle$, $V=\overline{\overline{V_G}}$, V_G – set of graph nodes (as centre of terrain squares, crossroads), A_G – set of graph arcs, $A_G \subset V_G \times V_G$, $A=\overline{\overline{A_G}}$. On each arc we have a defined value $d_{n,n'}$ of function d, which describes the terrain distance between the graph nodes n and n'. K objects (columns, trucks, tasks) move from source nodes vector $s=(s_1, s_2,\ldots, s_K)$ to destination nodes vector $t=(t_1, t_2,\ldots, t_K)$ of G. For further discussion we accepted the following notations (similar notations have been given in chapter 3.3.3.1, $s_k \equiv i^s(k)$, $t_k \equiv i^d(k)$):

$$I_k(s_k,t_k)=I_k = \left(\, i^0(k)=s_k,\ i^1(k),\ldots,\ i^r(k),\ldots,\ i^{R_k}(k)=t_k \right) \tag{4.1}$$

$$T_k(I_k)=T_k=\left(\tau^0(k),\tau^1(k),\ldots,\tau^r(k),\ldots,\tau^{R_k}(k)=\tau(I_k)\right) \tag{4.2}$$

$$V_k(I_k)=V_k=\left(v^k_{i^0(k),i^1(k)}, v^k_{i^1(k),i^2(k)},\ldots,\ v^k_{i^{R_k-1}(k),i^{R_k}(k)}\right) \tag{4.3}$$

where I_k – vector of nodes describing the path for the k-th object, $\underset{m\in\{1,\ldots,R_k\}}{\forall}\left(i^{m-1}(k),i^m(k)\right)\in A_G$; $i^r(k)$ – the r-th node on the path for the k-th object; s_k, t_k – source and destination nodes for the k-th object; T_k – vector of time instances of achieving the nodes belonging to the path for the k-th object; $\tau^r(k)$ – time instance of achieving node $i^r(k)$ by the head of the k-th object, $\underset{k=\overline{1,K}}{\forall}\ \underset{r=\overline{0,R_k-1}}{\forall}\ \tau^{r+1}(k)\geq\tau^r(k)\geq 0$ and $\underset{k=\overline{1,K}}{\forall}\ \tau^0(k)=0$; $\tau^{R_k}(k)=\tau(I_k)$ – time of achieving destination node by the k-th object; V_k – vector of velocities $v^k_{i^r(k),i^{r+1}(k)}$ of the k-th object on the arc $\left(i^r(k),i^{r+1}(k)\right)$ of its path; R_k – number of arcs belonging to the path of the k-th object. For the set

$\Pi(s,t)$ describing the set of vectors $I(s,t)$ of paths from $s=(s_1, s_2,\ldots,s_K)$ to $t=(t_1, t_2,\ldots,t_K)$ we have defined time τ^* as the earliest time of achieving the destination node by the most delayed object:

$$\tau^* = \min_{I(s,t)=(I_1,I_2,\ldots,I_K)\in\Pi(s,t)} \max_{k\in\{1,\ldots,K\}} \tau(I_k) \tag{4.4}$$

Let k^* denote the index of the object for which the moment of achieving the destination node for its path is the latest among paths for other objects, i.e. $k = k^* \Leftrightarrow \tau^{R_{k^*}}(k^*) = \max_{k\in\{1,\ldots,K\}} \tau^{R_k}(k)$. Let

$$IP_k = \left(i_1(k),\ i_2(k),\ldots,\ i_p(k),\ldots,\ i_{P_k}(k)\right) \tag{4.5}$$

denote a vector of nodes (checkpoints) at which we must align the head of the k-th object in relation to the heads of other objects, where $i_p(k)$ – the p-th element of IP_k satisfying: $\forall p=\overline{1,P_k}\ \exists r\in\{1,\ldots,R_k\}\ \ i_p(k)=i^r(k)$ and $r_p(k)=r\in\{1,\ldots,R_k\} \Leftrightarrow i_p(k)=i^r(k)$. The form of IP_k and $r_p(k)$ indicate that the path for the k-th object must cross by nodes belonging to IP_k. Let, by analogy

$$TP_k = \left(\tau_1(k),\ \tau_2(k),\ldots,\ \tau_p(k),\ldots,\ \tau_{P_k}(k)\right) \tag{4.6}$$

denote ordered set of time instances of achievement particular alignment nodes from set IP_k by the k-th object head, $\tau_p(k)$ denotes moment of achieving the p-th alignment node by the k-th object,

$$\tau_p(k) = \tau^0(k) + \sum_{r\in\{0,\ldots,r_p(k)-1\}} c_{i^r(k),i^{r+1}(k)} \tag{4.7}$$

where: $$c_{i^r(k),i^{r+1}(k)} = \frac{d_{i^r(k),i^{r+1}(k)}}{v^k_{i^r(k),i^{r+1}(k)}} \tag{4.8}$$

describes real movement time (time-cost) of the k-th object on the arc $\left(i^r(k), i^{r+1}(k)\right)\in A_G$ between $i^r(k)$ and $i^{r+1}(k)$ nodes of its path.

Additionally, we made the assumption that $P_1=P_2=\ldots=P_K=N$, i.e. for all objects exist the same number of alignment points (nodes). Let us define for each $p=1,\ldots,N$ the following characteristics:

$$\tau_p^{\max} = \max_{k\in\{1,\ldots,K\}} \tau_p(k) \tag{4.9}$$

$$\tau_p^{avg} = \frac{1}{K}\sum_{k=1}^{K} \tau_p(k) \tag{4.10}$$

$$\Delta\tau_p^*(k) = \tau_p(k^*) - \tau_p(k) \tag{4.11}$$

$$\Delta\tau_p^* = \max_{k\in\{1,\dots,K\}} \Delta\tau_p^*(k) \tag{4.12}$$

The most important criteria for movement synchronization scheduling can be divided into two categories. The first category is time of movement of *K* objects. We can define two basic measures of this category:

$$\text{(C.1.1):}\quad \tau^{\max} = \max_{k\in\{1,\dots,K\}} \tau^{R_k}(k) \to \min \tag{4.13}$$

$$\text{(C.1.2):}\quad \sum_{k=1}^{K} \tau^{R_k}(k) \to \min \tag{4.14}$$

The second category is "distance" between times of achieving alignment points by all of *K* objects. We can define four main measures of this category:

$$\text{(C.2.1):}\quad \sum_{p=1}^{N}\sum_{k=1}^{K}\left(\tau_p^{\max} - \tau_p(k)\right) \to \min \tag{4.15}$$

$$\text{(C.2.2):}\quad \min_{p\in\{1,\dots,N\}} \max_{k\in\{1,\dots,K\}} \left(\tau_p^{\max} - \tau_p(k)\right) \to \min \tag{4.16}$$

$$\text{(C.2.3):}\quad \sum_{k=1}^{K}\sum_{p=1}^{N}\left|\tau_p^{avg} - \tau_p(k)\right| \to \min \tag{4.17}$$

$$\text{(C.2.4):}\quad \min_{p\in\{1,\dots,N\}} \max_{k\in\{1,\dots,K\}} \left|\tau_p^{avg} - \tau_p(k)\right| \to \min \tag{4.18}$$

Presented criteria have the following interpretation: C.1.1 minimizes the time of achieving destination node by the last object (the most delayed); C.1.2 minimizes the total time of achieving destination nodes by all objects; C.2.1 minimizes total differences in times of achieving all checkpoints by all objects; C.2.2 minimizes the minimal of maximal differences in times of achieving any checkpoint by any object; C.2.3 minimizes total average differences in times of achieving all checkpoints by all objects; C.2.4 minimizes the minimal of maximal average differences of achieving any checkpoint by any object.

4.2.1.2. Formulation of movement synchronization problem with time (*MSST*)

One of the formulations of the optimization problem for movement synchronization of *K* objects can be defined as follows (we use criteria C.2.1 defined by (4.15)): for fixed paths I_k of each *k*-th object to determine such $v^k_{i^r(k),i^{r+1}(k)}$, $r=\overline{0,R_k-1}$, $k=\overline{1,K}$ that

$$\sum_{p=1}^{N}\sum_{k=1}^{K}\left(\tau_p^{\max} - \tau_p(k)\right) \to \min \tag{4.19}$$

with constraints:

$$v^k_{i^r(k),i^{r+1}(k)} \le v^{\max}_{i^r(k),i^{r+1}(k)}(k), \qquad r=\overline{0,R_k-1},\ \ k=\overline{1,K} \tag{4.20}$$

$$v^k_{i^r(k),i^{r+1}(k)} > 0, \qquad r=\overline{0,R_k-1},\ \ k=\overline{1,K} \tag{4.21}$$

where $v^{\max}_{i^r(k),i^{r+1}(k)}(k)$ describes the maximal velocity of the *k*-th object resulting from its technical properties and topographical condition on the arc $\left(i^r(k),i^{r+1}(k)\right)\in A_G$. Taking into consideration (4.7) and (4.9) we can write (4.19) as follows:

$$\sum_{p=1}^{N}\sum_{k=1}^{K}\left(\max_{j\in\{1,\dots,K\}}\left(\tau^0(j)+\sum_{\substack{r\in\{0,\dots,R_j-1\}\\ r\le r_p(j)}}\frac{d_{i^r(j),i^{r+1}(j)}}{v^k_{i^r(j),i^{r+1}(j)}}\right)-\left(\tau^0(k)+\sum_{\substack{r\in\{0,\dots,R_k-1\}\\ r\le r_p(k)}}\frac{d_{i^r(k),i^{r+1}(k)}}{v^k_{i^r(k),i^{r+1}(k)}}\right)\right)\to\min \tag{4.22}$$

Path *I_k* for the *k*-th object may be disjoint or not and must cross at fixed alignment points or we have to dynamically determine these points (e.g. during movement simulation/realization). In the first case we have an NP-hard optimization problem and we can solve it using approximation algorithms for finding disjoint paths (see chapter 3.4). In the second case we can use a two-stage approach: (*) finding the best paths for *K* objects iteratively using methods for finding the *m*-th (1st, 2nd, 3rd, etc.) best path for each of the *K* objects (Eppstein, 1999) and visiting specified nodes (Ibaraki, 1973; Ibaraki *et al.*, 1978); (**) synchronizing movement of *K* objects by solving problem (4.19)-(4.21) and using algorithms described in chapter 4.2.2 (Tarapata, 2008d; 2009a).
The multicriteria approach to movement synchronization scheduling is considered in chapter 4.3.

We can consider one of the extensions of problem (4.19)-(4.21): adding a constraint as follows

$$\tau^0(k)+\sum_{r\in\{0,\dots,R_k-1\}}\frac{d_{i^r(k),i^{r+1}(k)}}{v^k_{i^r(k),i^{r+1}(k)}}\le T^{\max}, \quad k=\overline{1,K} \tag{4.23}$$

we would like to find such a movement schedule that achieving the earliest moment of destination node by the latest object is no greater than $T^{\max}\ge\tau^*$.

To solve the problem (4.19)-(4.21) with the additional constraint (4.23), in generality, we define this problem in its changed form: for fixed paths I_k of each *k*-th object to determine such $x_{k,p}$, k=1,…,K, p=1,…., N that:

$$\sum_{p=1}^{N}\sum_{k=1}^{K}\left(\max_{j\in\{1,\dots,K\}}\left(\tau_p(j)+\sum_{i=1}^{p}x_{ji}\right)-\left(\tau_p(k)+\sum_{i=1}^{p}x_{ki}\right)\right)\to\min \tag{4.24}$$

with constraints:

$$\sum_{p=1}^{N} x_{kp} \le FT(k), \qquad k=1,...,K \tag{4.25}$$

$$x_{kp} \ge 0, \qquad k=1,...,K, \; p=1,...,N \tag{4.26}$$

where x_{kp} describes the time instance which is added to $\tau_p(k)$ for the k-th object in its p-th alignment point (node). It can be observed that $\tau_p(k)+\sum_{i=1}^{p} x_{ki} = \tau_p^{'}(k)$ which can be used in algorithms in chapter 4.2.2 as a modified (by algorithms) moment of achieving the p-th alignment point by the k-th object. Therefore, if we denote $\Delta\tau_p^{'\max}(k) = \tau_p^{'\max} - \tau_p^{'}(k)$, where $\tau_p^{'\max}$ is defined like in (4.9), then function (4.24) has an equivalent form of $\sum_{p=1}^{N}\sum_{k=1}^{K} \Delta\tau_p^{'\max}(k) \to \min$ and we obtain (4.19). Free time $FT(k)$ for the k-th object we define as: $FT(k) = T^{\max} - \tau^{R_k}(k)$.

We can observe that problem (4.19)-(4.21) is similar to a problem of task scheduling on parallel processors (Leung, 2004). The following similarities exist: (a) scheduling the problem before critical lines to minimize the sum of maximal delays in alignment points (nodes); the p-th critical line is created by nodes $i_p(1), i_p(2), ..., i_p(K)$; (b) we have parts of the path (arcs) as tasks; (c) we have moved objects as processors (K); (d) tasks are indivisible and dependent (the dependence is defined by each of the arc $\forall_{m\in\{1,...,R_k\}} \left(i^{m-1}(k), i^{m}(k)\right) \in A_G$ belonging to the path for each of the object). Differences: (a) tasks (arcs of the path) are assigned to processors (objects) (we have no influence on this assignment) and we decide only on the delays of the operation of processors (to increase realization time of tasks).

4.2.1.3. Formulation of movement synchronization problem with a group pattern (*MSSD*)

In chapter 4.2.1.2 we have defined movement synchronization of many objects with time (*MSST*): synchronization has been done considering achievement times of checkpoints. Here, we consider movement synchronization using some group patterns: in this case synchronization will be done according to some movement patterns and taking into account keeping terrain distances between objects resulting from a pattern. To define the *MSSD* problem we give some definitions.

As a group pattern (j-th) *of the* K objects numbered from 0 to K–1 we understand the following 2K-dimensional vector:

$$\left(x_0, y_0, \Delta x_1^j, \Delta y_1^j, ..., \Delta x_{K-1}^j, \Delta y_{K-1}^j\right) \tag{4.27}$$

where x_0, y_0 describe coordinates of the reference object (e.g. vehicle of commander).
With reference to this object we can set the location of the other objects in the group. The pairs $\left(\Delta x_k^j, \Delta y_k^j\right)$, $k = \overline{1, K-1}$ allow us to set coordinates of the *k*-th object inside the *j*-th group pattern as follows:

$$\left(x_k^j, y_k^j\right) = \left(x_0 + \Delta x_k^j, y_0 + \Delta y_k^j\right) \tag{4.28}$$

Additionally, we assume that there exists some a tolerance range δ^j for values Δx_k^j, Δy_k^j, $k = \overline{1, K-1}$. It means that coordinates of the *k*-th object in the *j*-th group pattern are defined as follows:

$$\left(x_k^j, y_k^j\right) = \left(x_0 + \Delta x_k^j \pm \delta^j, y_0 + \Delta y_k^j \pm \delta^j\right) \tag{4.29}$$

If coordinates of each object in a group satisfy (4.29) then we assume that the *j*-th group pattern is kept. It is important to say that a group pattern (4.27) is defined under the assumption that an angle α between the direction vector of the group and axis *0y* in the basic coordinate system is equal 0°. Hence, coordinates (4.28) and (4.29) are determined using this assumption. Relation between coordinates in the basic system *0xy* and rotated *0XY* with α angle is presented in (4.31). Examples of typical movement group patterns are presented in Fig. 4.1. It has been assumed that α=0°, that is the direction vector of the group cover *0y* axis of the basic coordinate system.

At the moment *t current location of group* is defined as follows:

$$\left(X_0(t), Y_0(t), X_1(t), Y_1(t), \dots, X_{K-1}(t), Y_{K-1}(t), \alpha\right) \tag{4.30}$$

where coordinates in (4.30) are determined in the coordinate system rotated with angle α with relation to the basic coordinate system and α describes the angle between the direction vector of the group and axis *0y* in the basic coordinate system. Relation between coordinates in the basic system *0xy* and rotated *0XY* with angle α is the following:

$$\left(x_k^j, y_k^j\right) = \left(X_k^j \cdot \cos\alpha - Y_k^j \cdot \sin\alpha, X_k^j \cdot \sin\alpha + Y_k^j \cdot \cos\alpha\right) \tag{4.31}$$

If we denote with $\left(x_0(t), y_0(t)\right)$ the location of the reference object at the moment *t* then the current, pattern location of the *K* considered objects grouped with *j*-th group pattern in the basic coordinate system at the moment *t* is defined as follows:

$$\left(x_0(t), y_0(t), x_1^j(t), y_1^j(t), \dots, x_{K-1}^j(t), y_{K-1}^j(t)\right) \tag{4.32}$$

where: $x_k^j(t) = x_0(t) + \Delta x_k^j$, $y_k^j(t) = y_0(t) + \Delta y_k^j$ describe the coordinate of the *k*-th object in the group according to the *j*-th pattern at the moment *t* in the basic coordinate system.

Since the "distance" $d^j(t)$ of the current group location from the *j*-th group pattern at the moment *t* we can understand the following function (with parameter *n*>0):

$$d^j(t) = \sum_{k=1}^{K-1} \left(q_{x_k}^n(t) + q_{y_k}^n(t) \right)^{\frac{1}{n}} \tag{4.33}$$

where:

$$q_{x_k}(t) = \begin{cases} x_k(t) - x_k^j(t), & \text{when } x_k(t) \notin [x_k^j(t) - \delta^j, x_k^j(t) + \delta^j] \\ 0 & , \text{ otherwise} \end{cases} \tag{4.34}$$

$$q_{y_k}(t) = \begin{cases} y_k(t) - y_k^j(t), & \text{when } y_k(t) \notin [y_k^j(t) - \delta^j, y_k^j(t) + \delta^j] \\ 0 & , \text{ otherwise} \end{cases} \tag{4.35}$$

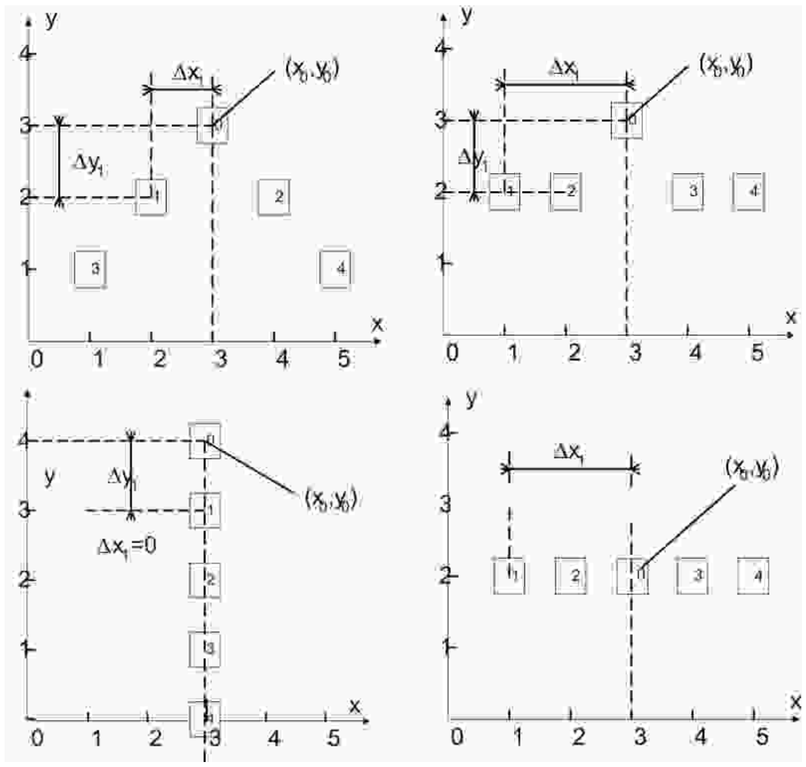


Fig. 4.1. Examples of typical movement group patterns for *K*=5 objects

We also assume that we have set, for each *k*-th object in the group, the movement path $I_k(s_k, t_k) = I_k = \left(i^0(k) = s_k,\ i^1(k), ...,\ i^r(k), ...,\ i^{R_k}(k) = t_k\right)$ described in (4.1) from the source node $i^0(k) = s_k$ to the destination node $i^{R_k}(k) = t_k$, $k = \overline{0, K-1}$ (apart from how can we determine these paths; we can use methods from chapter 3). Moreover, we assume that these paths assure us a satisfying condition, which concerns with distances $\Delta x_k^j \pm \delta^j$ (see (4.29)) from group pattern, for each *k*-th object in the group.

In the considered problem we want to set the movement speed for each object in the group in such a way, to minimize the total terrain distances from the group pattern in such moments when the reference object achieves each node on its path. This problem is defined in detail as follows: we want to find such values of speed $v_{i^r(k),i^{r+1}(k)}^k > 0$ for all objects on each arc $\left(i^r(k), i^{r+1}(k)\right)$ of the path I_k, $r = \overline{1, R_k - 1}$, $k = \overline{0, K-1}$, to minimize the value of the distance from group pattern defined as below (*MSSD* problem):

$$\sum_{p=1}^{R_0} d^j(t_p) \rightarrow \min \tag{4.36}$$

with constraints: (4.20) and (4.21)

where t_p denotes achieving the moment of the *p*-th node on the path for the reference object (with number *k*=0),

$$t_p = \sum_{r=1}^{p-1} c_{i^r(0),i^{r+1}(0)} \tag{4.37}$$

$$d^j(t_p) = \sum_{k=1}^{K-1} | q_{x_k}(t_p) | + | q_{y_k}(t_p) | \tag{4.38}$$

and $c_{i^r(k),i^{r+1}(k)}$ defined by (4.8), $q_{x_k}(t = t_p)$ defined by (4.34), $q_{y_k}(t = t_p)$ defined by (4.35).

Let us denote with $x_{i^p(0)}(t_p)$, $y_{i^p(0)}(t_p)$ coordinates of the reference object (with number 0) in the *p*-th node $i^p(0)$ on its path at the moment t_p. Pattern coordinates of the *k*-th object in the group according to the *j*-th pattern we calculate as follows:

$$x_k^j(t_p) = x_{i^p(0)}(t_p) + \Delta x_k^j \tag{4.39}$$

$$y_k^j(t_p) = y_{i^p(0)}(t_p) + \Delta y_k^j \tag{4.40}$$

Coordinates $x_k(t_p)$, $y_k(t_p)$ of the current location of the *k*-th object in the group at the moment t_p we calculate according to the following rule. First, we must determine between which nodes on the path the k–th object is located at moment t_p. We notice that the *k*-th object at the moment t_p is located between nodes on its path with such numbers r_k^* and $r_k^* + 1$ for which the following formula is fulfilled:

$$t_{r_k^*} = \sum_{r=2}^{r_k^*} c_{i^{r-1}(k),i^r(k)} < t_p \quad \text{and} \quad t_{r_k^*+1} = \sum_{r=2}^{r_k^*+1} c_{i^{r-1}(k),i^r(k)} > t_p \tag{4.41}$$

If $t_{r_k^*} = \sum_{r=2}^{r_k^*} c_{i^{r-1}(k),i^r(k)} = t_p$, then the *i*-th object is located inside the node of its path with number r_i^* at the moment t_p. Then, coordinates of the current location of the *i*-th object at the moment t_p are the following: $x_k(t_p) = x(i^{r_k^*}(k)), y_k(t_p) = y(i^{r_k^*}(k))$, where $x(i^{r_k^*}(k))$, $y(i^{r_k^*}(k))$ denote coordinates of node $i^{r_k^*}(k)$, in which the *i*-th object is located. Otherwise, when the *i*-th object is located between nodes with numbers r_k^* and r_k^*+1, the coordinates of the current location of the object are set according to the procedure described in Fig. 4.2. In this figure *dist* denotes the distance covered in the time of $t_p - t_{r_k^*}$ with the *k*-th object moving from node $i^{r_k^*}(k)$ to node $i^{r_k^*+1}(k)$. This distance is calculated from the following formula:

$$dist = v^k_{i^{r_k^*}(k),\, i^{r_k^*+1}(k)} \cdot (t_p - t_{r_k^*}) \tag{4.42}$$

where $v^k_{i^{r_k^*}(k),\, i^{r_k^*+1}(k)}$ denotes the speed of the *k*-th object between nodes $i^{r_k^*}(k)$ and $i^{r_k^*+1}(k)$.

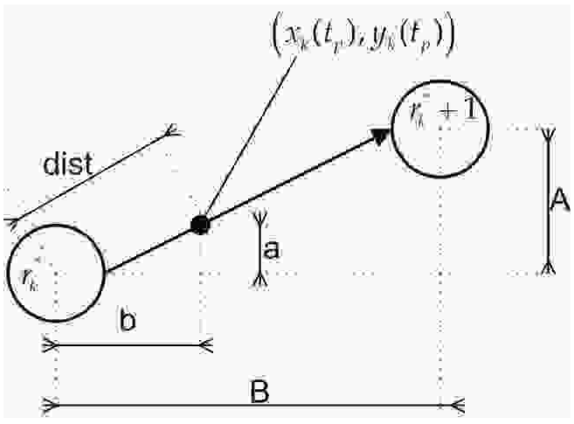


Fig. 4.2. Coordinates $(x_k(t_p), y_k(t_p))$ determining

Having *dist* we can calculate *a* and *b* from the system of equations:

$$\begin{cases} dist^2 = a^2 + b^2 \\ \dfrac{A}{B} = \dfrac{a}{b} \end{cases} \tag{4.43}$$

where $A = |\, y(i^{r_k^*+1}(k)) - y(i^{r_k^*}(k)) \,|$, $B = |\, x(i^{r_k^*+1}(k)) - x(i^{r_k^*}(k)) \,|$. We obtain:

$$b=\begin{cases}\sqrt{\dfrac{dist^2}{\dfrac{A^2}{B^2}+1}}, & \text{when } B\neq 0\\ 0 \quad , & \text{when } B=0\end{cases} \tag{4.44}$$

$$a=\begin{cases}\dfrac{A\cdot b}{B}, & \text{when } B\neq 0\\ dist \ , & \text{when } B=0\end{cases} \tag{4.45}$$

Coordinates of the current location of the *i*-th object at the moment t_p are used in equations (4.34), (4.35) and we can calculate them as follows:

$$x_k(t_p)=\begin{cases}x(i^{r_k^*}(k))+b, & \text{when } x(i^{r_k^*}(k))<x(i^{r_k^*+1}(k))\\ x(i^{r_k^*}(k))-b, & \text{when } x(i^{r_k^*}(k))\geq x(i^{r_k^*+1}(k))\end{cases} \tag{4.46}$$

$$y_k(t_p)=\begin{cases}x(i^{r_k^*}(k))+a, & \text{when } y(i^{r_k^*}(k))<y(i^{r_k^*+1}(k))\\ x(i^{r_k^*}(k))-a, & \text{when } y(i^{r_k^*}(k))\geq y(i^{r_k^*+1}(k))\end{cases} \tag{4.47}$$

Problem (4.36) with constrains (4.20) and (4.21) is a nonlinear programming problem and may be solved using one of commercial optimization packages (GAMS, MATHEMATICA) by invoking appropriate functions.

Another approach to define a group pattern has been presented in (Tarapata, 2007b). In this paper a multicriteria weighted graph similarity method for structural patterns recognition has been described. This approach may also be used for planning group movement with group patterns.

4.2.1.4. Example of the GAMS model for the *MSST* problem

The source code of the GAMS model for solving the *MSST* problem with parameters defined in Table 4.3 (for *FT*(*k*) in Table 4.4, chapter 4.2.2.3) is presented below. We set the following equivalence between notations being used in the *MSST* model and in the source code of the GAMS model (notation $x\equiv y$ describes that *x* in the GAMS model is equivalent to *y* in the *MSST* model):

tau(k,p) $\equiv \tau_p(k)$, FT(k) $\equiv FT(k)$, c(k,p) $\equiv \Delta\tau_p^{\max}(k)=\tau_p^{\max}-\tau_p(k)$, x(k,p) $\equiv x_{kp}$,

cost_p(k,p) $\equiv \sum_{i=1}^{p} x_{ki}$, max(p) $\equiv \max_{j\in\{1,\dots,K\}}\left(\tau_p(j)+\sum_{i=1}^{p} x_{ji}\right)$, z $\equiv$ value of objective function.

```
Sets
k                   objects for movement
/ 1, 2, 3 /
p                   alignment points (checkpoints)
/ 1, 2, 3, 4 / ;

Alias (p, pp)
Alias (k, kk)

Table
tau(k,p)             table of values from (4.5)
           1       2      3     4
      3    2       13     16    17
      2    5       9      13    16
      1    7       12     14    15;

Parameter
FT(k)              free times
/    3          1
     2          2
     1          0  /

Parameter
c(k,p)             table delta tau max p(k);

c(k,p)=smax(kk,tau(kk,p))-tau(k,p);
Variables
x(k,p)               decision variable in (4.24)-(4.26)
cost_p(k,p)          partial sum of x(k,p) from (4.24)
max(p)               the first component of sum from (4.24)
z                    value of objective function (4.24);

Positive Variable x;

Equations
partial_cost(k,p)    partial sum of x(k,p) from (4.24)
max_eq(p)            the first component of sum from (4.24)
FT_constr(k)         the k-th inequality from (4.25)
objective            value of objective function (4.24);

partial_cost(k,p)..cost_p(k,p)=e=sum(pp$(ORD(pp)le ORD(p)),x(k,pp));
max_eq(p) .. max(p)=e=smax(k,tau(k,p)+cost_p(k,p));
FT_constr(k) .. sum(p, x(k,p)) =l=  FT(k);
objective .. z  =e= sum((k,p),max(p)- (tau(k,p)+cost_p(k,p)));

Model Schedule /all/ ;

Solve Schedule using dnlp minimizing z ;

Display x.l, z.l;
```

Solving this problem using the GAMS/CONOPT solver we obtain:

```
----     61 VARIABLE x.L  decision variable in (4.24)-(4.26)

            1           4
2       2.000
3                   1.000

----     61 VARIABLE z.L                        =       14.000  value
                                           of objective function 4.24).
```

The obtained result is as follows: $x(2,1) = 2$, $x(3,4) = 1$ and remaining values of x(k,p) are equal to 0. The value of the objective function is equal to 14.

4.2.2. Scheduling Algorithms for Movement Synchronization

For solving the *MSS* problem two movement scheduling algorithms are presented: the first (*MSA*.1) is for solving problem (4.19)-(4.21) and the second (*MSA.2*) is for solving problem (4.24)-(4.26). Let us denote with $\tau_p^{'}(k)$, as it has been written in chapter 4.2.1.2, modified (by algorithms) the moment of achieving the *p*-th alignment point by the *k*-th object and $\Delta\tau_p^{'}(k) = \tau_p^{'}(k) - \tau_p(k)$.

4.2.2.1. Dynamic programming algorithm

The first algorithm *MSA*.1 is based on the dynamic programming approach.

Algorithm MSA.1

```
For each $p \in \{1,...,N\}$ recurrently compute the modified moments of
achieving alignment nodes for $K$ objects:

    $\tau_p^{'}(k) = \max_{j\in\{1,\ldots,K\}} \left(\Delta\tau_{p-1}^{'}(j) + \tau_p(j)\right), \quad \text{for } 1 \le k \le K$    (4.48)

and in addition $\tau_0^{'}(k) = \tau_0(k) = \tau^0(k)$, $1 \le k \le K$ .
```

Let us note that $\underset{k\in\{1,\ldots,K\}}{\forall} \Delta\tau_p^{'}(k) \ge 0$. It results from (4.48) and from the assumption that $\underset{k=\overline{1,K}}{\forall}\ \underset{r=\overline{0,R_k-1}}{\forall} \tau^{r+1}(k) > \tau^r(k) \ge 0$. Having $\underset{p\in\{1,\ldots,N\}}{\forall}\ \underset{k\in\{1,\ldots,K\}}{\forall} \tau_p^{'}(k)$ and $\Delta\tau_p^{'}(k)$, we can compute as follows: $\underset{k\in\{1,\ldots,K\}}{\forall}\ \underset{r\in\{0,\ldots,R_k\}}{\forall} \tau^{'r}(k) := \tau^r(k) + \Delta\tau_{q(r)}^{'}(k)$, $q(r) = \max\{p \in \{1,\ldots,N\} : r_p(k) \le r\}$ and $\underset{k\in\{1,\ldots,K\}}{\forall}\ \underset{r\in\{0,\ldots,R_k\}}{\forall} v_{i^r(k),i^{r+1}(k)}^{'k} := \dfrac{d_{i^r(k),i^{r+1}(k)}}{\tau^{'r+1}(k) - \tau^{'r}(k)}$. The complexity of the *MSA.1* algorithm is equal to $\Theta(K^2N)$ but we can obtain complexity $\Theta(KN)$ because for each $p \in \{1,\ldots,N\}$ $\tau_p^{'}(1) = \tau_p^{'}(2) = \ldots = \tau_p^{'}(K)$.

The idea of the algorithm is presented in Fig. 4.3 and the values of some characteristics in Table 4.1.

Table 4.1. Values of $\tau_p(k)$ and $\Delta\tau_p^{*}(k)$ for data from Fig. 4.3

k	$\tau_p(k)$				$\Delta\tau_p^{*}(k)$			
	p=1	*p*=2	*p*=3	*p*=4	*p*=1	*p*=2	*p*=3	*p*=4
3	2	13	16	17	5	-1	-2	-2
2	5	9	13	16	2	3	1	-1
1	7	12	14	15	0	0	0	0

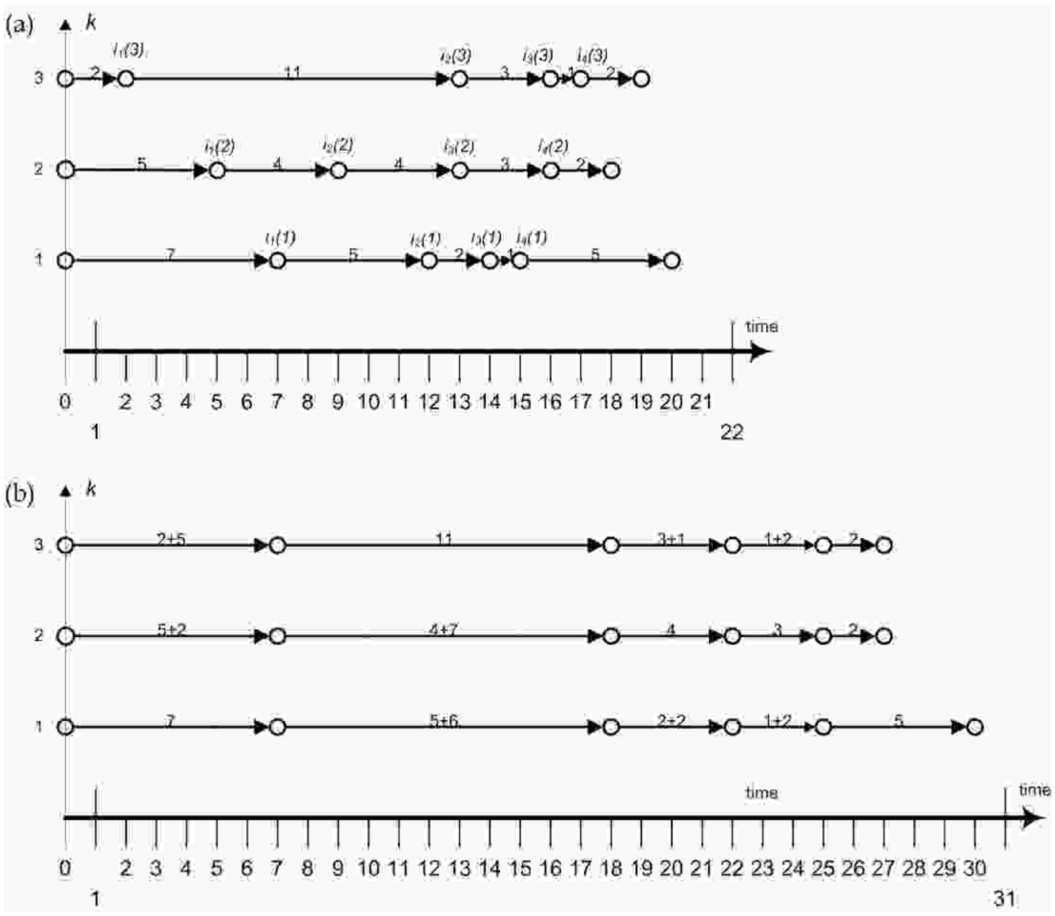


Fig. 4.3. An idea of the *MSA*.1 algorithm: (a) paths for *K*=3 objects with times of achieving alignment nodes $i_p(k)$ on paths for each *p*=1,…4 and *k*=1,…,3; (b) the result of algorithm *MSA*.1 – times of achieving alignment nodes by *K* objects have the same value equalling 25

Theorem 4.1

Algorithm MSA.1 solves the problem (4.19)-(4.21) *optimally.*

Proof:

For fixed *p*∈{1,...,*N*} the following condition is fulfilled: $\tau_p^{'}(1)=\tau_p^{'}(2)=...=\tau_p^{'}(K)$, hence $\sum_{k=1}^{K}\left(\tau_p^{'\max}-\tau_p^{'}(k)\right)=0$, because the condition is fulfilled for each *p*∈{1,...,*N*}, so we obtain: $\sum_{p=1}^{N}\sum_{k=1}^{K}\left(\tau_p^{'\max}-\tau_p^{'}(k)\right)=0$.

♦

It is easy to notice, that *MSA*.1 algorithm simultaneously minimizes criteria C.2.2, C.2.3 and C.2.4.

Let us note that if two alignment nodes p and p+1 are neighbouring nodes on a path for the k-th object, that is the following formula is fulfilled $r_p(k) = i^r(k) \Rightarrow r_{p+1}(k) = i^{r+1}(k)$ and then from (4.8) and (4.48) results that:

$$\begin{aligned} \tau'_p(k) &= \max_{l \in \{1,\dots,K\}} \left(\Delta\tau'_{p-1}(l) + \tau_p(l) \right) = \max_{l \in \{1,\dots,K\}} \left(\tau'_{p-1}(l) - \tau_{p-1}(l) + \tau_p(l) \right) = \\ &= \max_{l \in \{1,\dots,K\}} \left(\tau'_{p-1}(l) - \tau_{p-1}(l) + \tau_{p-1}(l) + c_{i^{r_{p-1}(l)}(l), i^{r_p(l)}(l)} \right) = \\ &= \max_{l \in \{1,\dots,K\}} \left(\tau'_{p-1}(l) + c_{i^{r_{p-1}(l)}(l), i^{r_p(l)}(l)} \right) = \tau'_{p-1}(k) + \max_{l \in \{1,\dots,K\}} c_{i^{r_{p-1}(l)}(l), i^{r_p(l)}(l)} \end{aligned} \tag{4.49}$$

where $c_{i^{r_{p-1}(l)}(l), i^{r_p(l)}(l)}$ is defined by (4.8).

Algorithm *MSA.1*, even though is very simple, has interesting properties (Theorem 4.2 and Theorem 4.3).

Theorem 4.2

Necessary conditions for obtaining, for each solution $\tau'_N(k)$ from MSA.1 algorithm, that:

$$\max_{k \in \{1,\dots,K\}} \tau'_N(k) \le \tau_N(k^*) \tag{4.50}$$

are following:

1°. $$\forall_{p \in \{1,\dots,N\}} \forall_{k \in \{1,\dots,K\}} \Delta\tau^*_p(k) \ge 0 \tag{4.51}$$

2°. $$\forall_{k \in \{1,\dots,K\}} \Delta\tau^*_1(k) \le \Delta\tau^*_2(k) \le \dots \le \Delta\tau^*_N(k) \tag{4.52}$$

Proof:

Ad.1°

Let us assume conversely, that

$$\exists_{p' \in \{1,\dots,N\}} \exists_{k' \in \{1,\dots,K\}} \Delta\tau^*_{p'}(k') < 0$$

Then from (4.11) results that $\tau_{p'}(k^*) < \tau_{p'}(k')$. But from (4.48) results that for each $k \in \{1,\dots,K\}$ the following equality is true: $\tau'_{p'}(k) = \max_{l \in \{1,\dots,K\}} \left(\Delta\tau'_{p'-1}(l) + \tau_{p'}(l) \right)$, because the following condition is fulfilled: $\forall_{l \in \{1,\dots,K\}} \Delta\tau'_{p'-1}(l) \ge 0$, hence $\tau'_{p'}(k') \ge \tau_{p'}(k') > \tau_{p'}(k^*)$. If we place k'=k* and p'=N, then we obtain that $\tau'_N(k^*) > \tau_N(k^*)$. This contradiction ends the first part of the proof.

Ad.2°

Let us assume conversely, that

$$\exists_{p' \in \{1,\dots,N-1\}} \exists_{k' \in \{1,\dots,K\}} \Delta\tau^*_{p'}(k') > \Delta\tau^*_{p'+1}(k') \tag{4.53}$$

and that the following conditions, resulting from the first part of the proof, are satisfied:

$$\underset{k\in\{1,\ldots,K\}}{\forall} \Delta\tau^{*}_{p'}(k) \geq 0, \quad \Delta\tau^{*}_{p'+1}(k) \geq 0 \tag{4.54}$$

We will show that if formula (4.53) is fulfilled, then $\tau'_{p'+1}(k^{*}) > \tau_{p'+1}(k^{*})$. Let us assume that p'=1. Then from (4.54) we have that: $\underset{k'\in\{1,\ldots,K\}}{\exists} \Delta\tau^{*}_{1}(k') > \Delta\tau^{*}_{2}(k')$ or equivalently

$$\underset{k'\in\{1,\ldots,K\}}{\exists} \tau_1(k^{*}) - \tau_1(k') > \tau_2(k^{*}) - \tau_2(k') \tag{4.55}$$

From (4.48) and (4.54) results that

$$\tau'_1(k') = \max_{k\in\{1,\ldots,K\}} \tau_1(k) = \tau_1(k^{*}) \tag{4.56}$$

$$\tau'_2(k') = \max_{k\in\{1,\ldots,K\}} \left(\tau'_1(k) - \tau_1(k) + \tau_2(k)\right) = \max_{k\in\{1,\ldots,K\}} \left(\tau_1(k^{*}) - \tau_1(k) + \tau_2(k)\right) \tag{4.57}$$

Taking into account (4.55) we obtain:

$$\tau_1(k^{*}) - \tau_1(k') + \tau_2(k') - \tau_2(k^{*}) > 0 \tag{4.58}$$

or equivalently

$$\tau_1(k^{*}) - \tau_1(k') + \tau_2(k') > \tau_2(k^{*}) \tag{4.59}$$

If we place (4.57) into (4.58) we obtain:

$$\tau'_2(k') = \max_{k\in\{1,\ldots,K\}} \left(\tau_1(k^{*}) - \tau_1(k) + \tau_2(k)\right) \geq \tau_1(k^{*}) - \tau_1(k') + \tau_2(k') > \tau_2(k^{*})$$

If we set $k'=k^{*}$, then $\tau'_2(k^{*}) > \tau_2(k^{*})$ and we obtain a contradiction with (4.50). Therefore we have proved that apart from (4.51), the condition (4.52) is necessary to fulfil (4.50).

♦

Theorem 4.3

Conditions (4.51) *and* (4.52) *are jointly sufficient to satisfy formula* (4.50) *for each solution* $\tau'_N(k)$ *obtained from algorithm MSA.1.*

Proof:

To prove the thesis of the theorem we need to show that if (4.51) and (4.52) are fulfilled then for each p=1,...,N

$$\tau'_p(k^{*}) \leq \tau_p(k^{*}) \tag{4.60}$$

We will prove it by induction on p. From (4.56) results that formula (4.60) is true for p=1. Let us place p=m and let us assume that (4.60) is true. We obtain

$$\tau_m^{'}(k^*) = \max_{l \in \{1,\ldots,K\}} \left(\Delta\tau_{m-1}^{'}(l) + \tau_m(l)\right) \le \tau_m(k^*) \tag{4.61}$$

We will show that formula (4.60) is true for m+1. We have

$$\tau_{m+1}^{'}(k^*) = \max_{l \in \{1,\ldots,K\}} \left(\Delta\tau_m^{'}(l) + \tau_{m+1}(l)\right) = \max_{l \in \{1,\ldots,K\}} \left(\tau_m^{'}(l) - \tau_m(l) + \tau_{m+1}(l)\right) \tag{4.62}$$

From formula (4.61) results that $\tau_m^{'}(k^*) \le \tau_m(k^*)$, from assumption (4.51) results that $\forall_{l \in \{1,\ldots,K\}} \tau_p(k^*) \ge \tau_p(l)$ and from assumption (4.52) that $\forall_{l \in \{1,\ldots,K\}} \tau_m(k^*) - \tau_m(l) \le \tau_{m+1}(k^*) - \tau_{m+1}(l)$, hence we can write (4.62) as follows:

$$\begin{aligned} \tau_{m+1}^{'}(k^*) &= \max_{l \in \{1,\ldots,K\}} \left(\tau_m^{'}(l) - \tau_m(l) + \tau_{m+1}(l)\right) \le \max_{l \in \{1,\ldots,K\}} \left(\tau_m(k^*) - \tau_m(l) + \tau_{m+1}(l)\right) \le \\ &\le \max_{l \in \{1,\ldots,K\}} \left(\tau_{m+1}(k^*) - \tau_{m+1}(l) + \tau_{m+1}(l)\right) \le \tau_{m+1}(k^*) \end{aligned}$$

Q.E.D.

♦

The main conclusion from Theorem 4.2 and Theorem 4.3 is as follows: if for each k=1,…,K we set $r_N(k)$=R_k then from (4.60) we have: $\max_{k \in \{1,\ldots,K\}} \tau^{'R_k}(k) \le \tau^{R_{k^*}}(k^*)$. It means that the value of τ^* has not changed, i.e. the latest (the most delayed) moment of achieving destination nodes by all objects have not changed, and then constraint (4.23) is fulfilled. It means that *MSA.1* optimally also solves problems (4.19)-(4.21) with constraint (4.23).

In Fig. 4.4 we present conclusions from Theorem 4.2 and Theorem 4.3. Table 4.2 presents some characteristics of the problem from Fig. 4.4.

Table 4.2. Values of $\tau_p(k)$ and $\Delta\tau_p^*(k)$ for data from Fig. 4.4

k	$\tau_p(k)$				$\Delta\tau_p^*(k)$			
	p=1	p=2	p=3	p=4	p=1	p=2	p=3	p=4
3	1	5	11	13	3	3	3	4
2	2	4	9	10	2	4	5	7
1	4	8	14	17	0	0	0	0

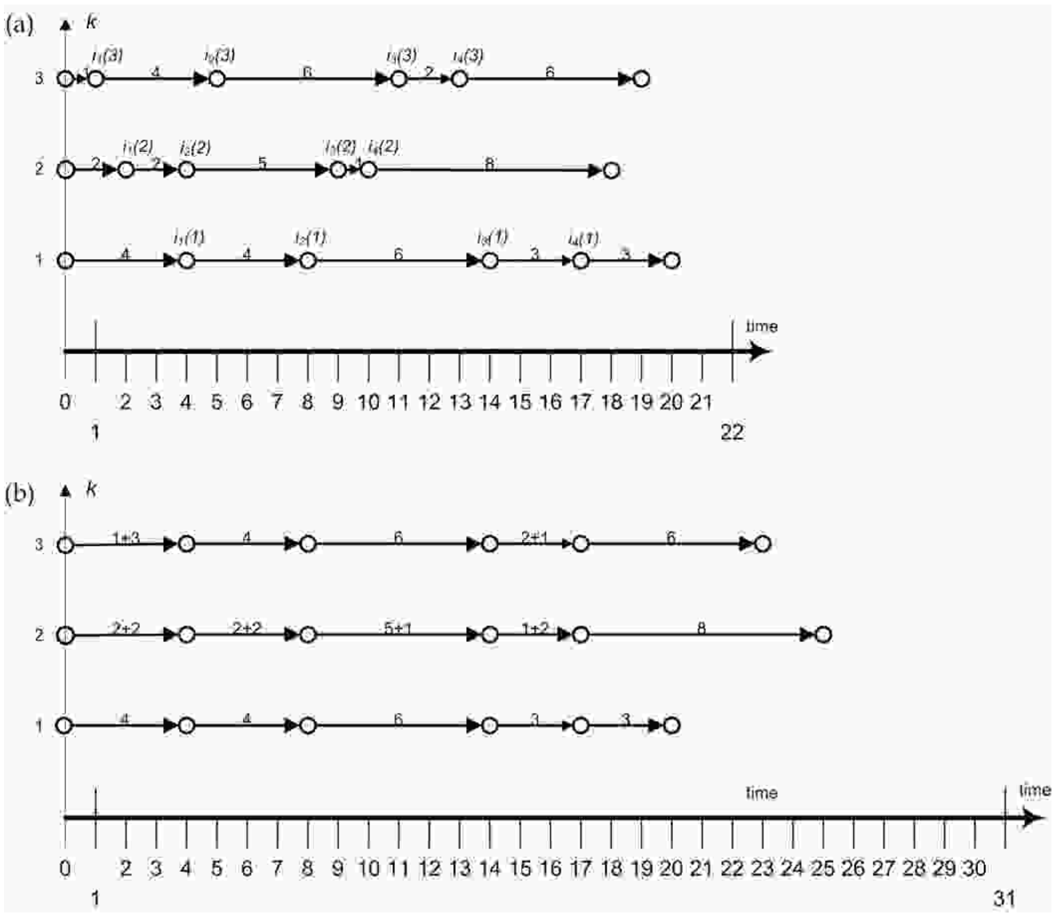


Fig. 4.4. Paths for K=3 objects satisfying conditions of Theorem 4.2 and Theorem 4.3: (a) times of achieving alignment nodes $i_p(k)$ on the paths for each p=1,…4 and k=1,…,3; (b) result of algorithm *MSA*.1 – times of achieving alignment nodes by K objects have the same value equalling 17) and they are not greater than for object k^*=1

Theorem 4.4

Let $\Delta\tau_p^d(k) = \max\{\Delta\tau_{p-1}^*(k) - \Delta\tau_p^*(k), 0\}$ *and* $\Delta\tau_p^d = \max_{k\in\{1,...,K\}} \Delta\tau_p^d(k)$ *be defined. If* $\forall_{p\in\{1,...,N\}} \forall_{k\in\{1,...,K\}} \Delta\tau_p^*(k) \geq 0$ *then the following formula is fulfilled:*

$$\forall_{p\in\{1,...,N\}} \tau_p^{'}(k^*) = \tau_p(k^*) + \sum_{\substack{p'\in\{1,...,N\} \\ p' \leq p}} \Delta\tau_{p'}^d \tag{4.63}$$

Proof:

Let us note that if $\forall_{k\in\{1,...,K\}} \Delta\tau_1^*(k) \leq \Delta\tau_2^*(k) \leq ... \leq \Delta\tau_N^*(k)$ (the fulfilment of the condition (4.52)) then the following formula is fulfilled :

$$\forall_{p\in\{1,...,N\}} \Delta\tau_p^d(k) = \max\{\Delta\tau_{p-1}^*(k) - \Delta\tau_p^*(k), 0\} = 0 \rightarrow \Delta\tau_p^d = 0$$

hence $\tau_p^{'}(k^*) = \tau_p(k^*)$.

To prove that formula (4.63) is true we will show, taking into account (4.48), that for each *p*=1,…,*N* the following formula is fulfilled:

$$\tau_p^{'}(k^*) = \tau_p(k^*) + \sum_{\substack{p'\in\{1,\dots,N\}\\ p'\le p}} \Delta\tau_{p'}^d = \max_{k\in\{1,\dots,K\}} \left(\Delta\tau_{p-1}^{'}(k) + \tau_p(k)\right) \tag{4.64}$$

We will prove, by induction on *p*, that the left side $L = \tau_p(k^*) + \sum_{\substack{p'\in\{1,\dots,N\}\\ p'\le p}} \Delta\tau_{p'}^d$ of the formula (4.64) is equal to the right side $R = \max_{k\in\{1,\dots,K\}} \left(\Delta\tau_{p-1}^{'}(k) + \tau_p(k)\right)$, that is *L*=*R*.

For *p*=1$\Rightarrow L = \tau_1(k^*) + \Delta\tau_1^d$, because $\Delta\tau_p^d(k) = \max\left\{\Delta\tau_{p-1}^*(k) - \Delta\tau_p^*(k), 0\right\}$, $\Delta\tau_p^d = \max_{k\in\{1,\dots,K\}} \Delta\tau_p^d(k)$ and from (4.11) results that $\Delta\tau_1^d(k) = \max\left\{\Delta\tau_0^*(k) - \Delta\tau_1^*(k), 0\right\} = 0$ for each *k*=1,…,*K*, hence $\Delta\tau_1^d = 0$ and $L = \tau_1(k^*)$.

The right side of the formula (4.64) is equal:

$$R = \max_{k\in\{1,\dots,K\}} \left(\Delta\tau_0^{'}(k) + \tau_1(k)\right) = \max_{k\in\{1,\dots,K\}} \left(\tau_0^{'}(k) - \tau_0(k) + \tau_1(k)\right) = \max_{k\in\{1,\dots,K\}} \tau_1(k) = \tau_1(k^*)$$

and we have obtained: *L*=*R*.

For *p*=2:

$$\begin{aligned} L &= \tau_2(k^*) + \Delta\tau_1^d + \Delta\tau_2^d = \tau_2(k^*) + \max_{k\in\{1,\dots,K\}} \Delta\tau_2^d(k) = \\ &= \tau_2(k^*) + \max_{k\in\{1,\dots,K\}} \left(\max\left\{\Delta\tau_1^*(k) - \Delta\tau_2^*(k), 0\right\}\right) = \\ &= \max_{k\in\{1,\dots,K\}} \left(\max\left\{\Delta\tau_1^*(k) - \Delta\tau_2^*(k), 0\right\} + \tau_2(k^*)\right) \end{aligned}$$

$$\begin{aligned} R &= \max_{k\in\{1,\dots,K\}} \left(\Delta\tau_1^{'}(k) + \tau_2(k)\right) = \max_{k\in\{1,\dots,K\}} \left(\tau_1^{'}(k) - \tau_1(k) + \tau_2(k)\right) = \\ &= \max_{k\in\{1,\dots,K\}} \left(\tau_1(k^*) - \tau_1(k) + \tau_2(k)\right) = \max_{k\in\{1,\dots,K\}} \left(\Delta\tau_1^*(k) + \tau_2(k)\right) \end{aligned}$$

From analysis of *L* and *R* we obtain, that to satisfy *L*=*R* it is required that $\exists_{l_1,l_2\in\{1,\dots,K\}}$ for which

$$\max\left\{\Delta\tau_1^*(l_1) - \Delta\tau_2^*(l_1), 0\right\} + \tau_2(k^*) = \Delta\tau_1^*(l_2) + \tau_2(l_2)$$

that is

$$\max\left\{\tau_1(k^*) - \tau_1(l_1) - \tau_2(k^*) + \tau_2(l_1) + \tau_2(k^*), \tau_2(k^*)\right\} = \Delta\tau_1^*(l_2) + \tau_2(l_2)$$

Hence

$$\max\left\{\tau_1(k^*) - \tau_1(l_1) + \tau_2(l_1), \tau_2(k^*)\right\} = \tau_1(k^*) - \tau_1(l_2) + \tau_2(l_2) \tag{4.65}$$

The equality (4.65) is always true because $\tau_1(k^*) - \tau_1(l_1) \ge 0$ (theorem assumption), hence $\tau_1(k^*) - \tau_1(l_1) + \tau_2(l_1) \ge \tau_2(k^*)$ and *L*=*R* for $l_1 = l_2$.

Let us set $p=m=2$ and we will prove that formula (4.63) is true for $m+1$. Let us note that

$$L=\tau_p^{'}(k^*)=\tau_p(k^*)+\sum_{\substack{p'\in\{1,\dots,N\}\\ p'\le p}}\Delta\tau_{p'}^{d}=\tau_{p-1}^{'}(k^*)+\Delta\tau_p^d-\tau_{p-1}(k^*)+\tau_p(k^*)$$

For $m+1$ we obtain:

$$L=\tau_m^{'}(k^*)+\Delta\tau_{m+1}^{d}-\tau_m(k^*)+\tau_{m+1}(k^*)$$

$$R=\max_{k\in\{1,\dots,K\}}\left(\Delta\tau_m^{'}(k)+\tau_{m+1}(k)\right)$$

and

$$L=\tau_m^{'}(k^*)+\max_{k\in\{1,\dots,K\}}\left(\max\left\{\Delta\tau_m^{*}(k)-\Delta\tau_{m+1}^{*}(k),0\right\}\right)-\tau_m(k^*)+\tau_{m+1}(k^*)=$$

$$=\max_{k\in\{1,\dots,K\}}\begin{pmatrix}\max\left\{\Delta\tau_m^{*}(k)-\Delta\tau_{m+1}^{*}(k)-\tau_m(k^*)+\tau_{m+1}(k^*)+\tau_m^{'}(k^*),\right.\\ \left.-\tau_m(k^*)+\tau_{m+1}(k^*)+\tau_m^{'}(k^*)\right\}\end{pmatrix}$$

To satisfy $L=R$ it is needed that $\exists_{l_1,l_2\in\{1,\dots,K\}}$ for which:

$$\max\left\{\Delta\tau_m^{*}(l_1)-\Delta\tau_{m+1}^{*}(l_1)-\tau_m(k^*)+\tau_{m+1}(k^*)+\tau_m^{'}(k^*),\right.$$
$$\left.-\tau_m(k^*)+\tau_{m+1}(k^*)+\tau_m^{'}(k^*)\right\}=\Delta\tau_m^{'}(l_2)+\tau_{m+1}(l_2)$$

that is

$$\max\left\{\tau_m(k^*)-\tau_m(l_1)-\tau_{m+1}(k^*)+\tau_{m+1}(l_1)-\tau_m(k^*)+\tau_{m+1}(k^*)+\tau_m^{'}(k^*),\right.$$
$$\left.-\tau_m(k^*)+\tau_{m+1}(k^*)+\tau_m^{'}(k^*)\right\}=\Delta\tau_m^{'}(l_2)+\tau_{m+1}(l_2)$$

Reducing this formula, we obtain:

$$\begin{aligned}&\max\left\{\tau_{m+1}(l_1)-\tau_m(l_1)+\tau_m^{'}(k^*),-\tau_m(k^*)+\tau_{m+1}(k^*)+\tau_m^{'}(k^*)\right\}=\\&=\tau_{m+1}(l_2)-\tau_m(l_2)+\tau_m^{'}(l_2)\end{aligned}\tag{4.66}$$

If we set $l_1=l_2=k^*$ then the equality (4.66) is fulfilled. The equality is fulfilled too, for any $l_1=l_2$ such that $\tau_{m+1}(l_1)-\tau_m(l_1)\ge\tau_{m+1}(k^*)-\tau_m(k^*)$.

♦

Conclusions from Theorem 4.4:

1°
$$\max_{k\in\{1,\dots,K\}}\tau^{'R_k}(k)=\tau_N^{'}(k^*)+\max_{k\in\{1,\dots,K\}}\left(\tau^{R_k}(k)-\tau_N(k)\right)=$$
$$=\tau_N(k^*)+\sum_{p\in\{1,\dots,N\}}\Delta\tau_p^d+\max_{k\in\{1,\dots,K\}}\left(\tau^{R_k}(k)-\tau_N(k)\right)$$

2° For each $T^{\max}\ge\tau_N(k^*)+\sum_{p\in\{1,\dots,N\}}\Delta\tau_p^d+\max_{k\in\{1,\dots,K\}}\left(\tau^{R_k}(k)-\tau_N(k)\right)$ the following formula is fulfilled: $\sum_{p=1}^{N}\sum_{k=1}^{K}\left(\tau_p^{'\max}-\tau_p^{'}(k)\right)=0$.

From conclusions 1° and 2° results that for each T^{max}, which satisfy condition 1° the following formula is true: $\max_{k\in\{1,\ldots,K\}} \tau^{'R_k}(k) \le T^{\max}$, that is τ^* has no greater value than T^{max} and simultaneously the following condition is fulfilled: $\sum_{p=1}^{N}\sum_{k=1}^{K}\left(\tau_p^{'\max} - \tau_p^{'}(k)\right) = 0$. We can check condition 2° in time of $\Theta(KN)$.

4.2.2.2. Cost-profit approximation algorithm

We can present the heuristic (greedy) algorithm *MSA.2*, which solves the problem (4.24)-(4.26) (it is equivalent to the problem (4.19)-(4.21) with constraint (4.23)). We define the notations used inside the algorithm: *card*(x) – cardinality of the set x; $a_p(k)$ – time instance which is added to $\tau_p(k)$. We also define three sets of checkpoints which satisfy some conditions:

$$P_s^+(k) = \left\{p \in \{s,\ldots,N\} : \Delta\tau_p^{\max}(k) > 0\right\} \tag{4.67}$$

$$P_s^{\ge}(k) = \left\{p \in \{s,\ldots,N\} : \Delta\tau_p^{\max}(k) - a_p(k) \ge 0\right\} \tag{4.68}$$

$$P_s^{<}(k) = \left\{p \in \{s,\ldots,N\} : \Delta\tau_p^{\max}(k) - a_p(k) < 0\right\} \tag{4.69}$$

Functions $Z(\cdot)$ and $L(\cdot)$ describe "profit" (Z) and "cost" (L) of decreasing $\Delta\tau_p^{\max}(k)$ with value $a_{s_k}(k)$, $s_k \in P_{s_k}^+(k)$:

$$Z(a_{s_k}(k)) = a_{s_k}(k)\cdot card\left(P_{s_k}^{\ge}(k)\right) + \sum_{p\in P_{s_k}^{<}(k)} \Delta\tau_p^{\max}(k) \tag{4.70}$$

$$L(a_{s_k}(k)) = (K-1)\cdot \sum_{p\in P_{s_k}^{<}(k)} \left|\Delta\tau_p^{\max}(k) - a_{s_k}(k)\right| \tag{4.71}$$

Value $x_{k,p} := x_{k,p} + a_p(k)$ (in step 10 of the *MSA*.2 algorithm) is equal to the sum of $a_p(k)$ values that are determined for all iterations of *MSA*.2 and for every k and p. The idea of the algorithm *MSA*.2 consists of decreasing the value of $OBJ = \sum_{p=1}^{N}\sum_{k=1}^{K}\Delta\tau_p^{'\max}(k)$ by decreasing the value of $\Delta\tau_p^{'\max}(k)$ for any k and p.

To set an examination order vector *KO* of *K* objects in the *MSA*.2 algorithm we use an object order *ObjOrder*∈{0,…,3} strategy (the 3rd step of the algorithm): *ObjOrder*=0 – set elements of *KO* iteratively, from k=1 to k=K; *ObjOrder*=1 – set elements of *KO* randomly, with uniform distribution on the set {1,…,K}; *ObjOrder*=2 – set elements of *KO* iteratively, starting from such a k, which corresponds to the first greatest, second greatest, …, the K-th greatest values of the coordinates of the vector *FT*; *ObjOrder*=3 – set elements of *KO* iteratively, starting from such a k which corresponds to the first smallest, second smallest,…, the K-th smallest values of the coordinates of the vector *FT*.

Algorithm MSA.2

```
Given sets: $I_k$, $T_k$, $IP_k$, $TP_k$ for each $k$=1,…,$K$ and values
ObjOrder, Strategy;
Initialize: $\forall_{k\in\{1,\dots,K\}}\forall_{p\in\{1,\dots,N\}} a_p(k):=0$; $\forall_{k\in\{1,\dots,K\}}\forall_{p\in\{1,\dots,N\}} x_{k,p}:=0$; counter:=N;
$\forall_{k\in\{1,\dots,K\}} FT(k):=\tau^{R_{k^*}}(k^*)-\tau^{R_k}(k)$; $\forall_{k\in\{1,\dots,K\}}\forall_{p\in\{1,\dots,N\}} \Delta\tau_p^{'\max}(k):=\tau_p^{\max}-\tau_p(k)$;
1.  WHILE $\left(\exists_{k\in\{1,\dots,K\}} FT(k)>0\right)\wedge(counter>0)$ DO
2.    counter:=0;
3.    To determine KO vector using ObjOrder;
3a.   FOR k=KO[1],…,KO[K] DO
4.      IF $FT(k)>0$ THEN
5.        Use current Strategy to find $s_k$ and $a_{s_k}(k)$;
6.        IF $a_{s_k}(k)>0$ THEN
7.          $\forall_{p\in\{s_k,\dots,N\}} \Delta\tau_p^{'\max}(k):=\Delta\tau_p^{'\max}(k)-a_{s_k}(k)$;
8.          $\forall_{p\in P_{s_k}^{<}(k)}\forall_{j\in\{1,\dots,K\}} \Delta\tau_p^{'\max}(j):=\Delta\tau_p^{'\max}(j)+\left|\Delta\tau_p^{'\max}(k)\right|$;
9.          $FT(k):=FT(k)-a_{s_k}(k)$;
10.         $x_{k,s_k}:=x_{k,s_k}+a_{s_k}(k)$;
11.         counter:=counter+1; $a_{s_k}(k):=0$;
12.       END IF;
13.     END IF;
14.   END FOR;
15. END WHILE.
```

To find values of $s_k\in P_1^+(k)$ and $a_{s_k}(k)\in\left(0,\ \min\left\{\Delta\tau_{s_k}^{'\max}(k),FT(k)\right\}\right]$ we use *Strategy*∈{0,…,4} (the 5th step of the algorithm): *Strategy*=0 – finds such a value s_k and maximal value $a_{s_k}(k)$ for which condition $Z(a_{s_k}(k))>L(a_{s_k}(k))$ is fulfilled; *Strategy*=1 – find such a value s_k and value $a_{s_k}(k)$ for which value $Z(a_{s_k}(k))-L(a_{s_k}(k))$ is maximal and positive; *Strategy*=2 – find such a value s_k N times and randomly $a_{s_k}(k)$ for which value $Z(a_{s_k}(k))-L(a_{s_k}(k))$ is maximal and positive; *Strategy*=3 – find N times randomly such values s_k and $a_{s_k}(k)$ for which the value $Z(a_{s_k}(k))-L(a_{s_k}(k))$ is maximal and positive; *Strategy*=4 – like for *Strategy*=3 but we draw values s_k and $a_{s_k}(k)$ only one time.

For example, when *ObjOrder*=0 and *Strategy*=0, the *OBJ* will be decreased when we select such a maximal value of $a_{s_k}(k)\in\left(0,\ \min\left\{\Delta\tau_{s_k}^{'\max}(k),FT(k)\right\}\right]$ for any $s_k\in P_1^+(k)$ that $Z(a_{s_k}(k))>L(a_{s_k}(k))$. Let us take into account the second row of Table 4.4 (for k=2). It is profitable to set a_1=2=min{max{2,4,3,1}, 2, 2}, because when we decrease values of $\Delta\tau_p^{\max}(2)$ for $p\in P_1^{\geq}(2)=\{1,\ 2,\ 3\}$ then our "profit" (decreasing the value of *OBJ*) is equal:

$$Z(a_1(2)) = a_1(2)\cdot card\left(P_1^{\geq}(2)\right) + \sum_{p\in P_1^{<}(2)} \Delta\tau_p^{\max}(2) = 2\cdot 3+1=7\,.$$

"Cost" is equal $L(a_1(2)) = (3-1)\cdot \sum_{p\in P_1^{<}(2)} \left|\Delta\tau_p^{\max}(2)-a_1(2)\right| = 2\cdot 1$ (increasing the value of *OBJ*). Afterwards, in steps 7-9 we decrease the value of $\Delta\tau_p^{'\max}(k)$ and $FT(k)$ with $a_{s_k}(k)$ for all $p\geq s_k$. In the case of $\Delta\tau_p^{\max}(k)-a_{s_k}(k)<0$ in step 7, we must increase this value like in step 8. The algorithm tries to decrease the value of *OBJ* until the free time $FT(k)$ for all k will be equal to zero or when $a_{s_k}(k)>0$ (for which the condition $Z(a_{s_k}(k))>L(a_{s_k}(k))$ is fulfilled) does not exist for any k and p (variable *counter*=0).

Let $\Delta\tau^{\min}(k)=\min_{p\in\{1,..,N\}}\left\{\tau_p^{\max}-\tau_p(k)\right\}$, if $\tau_p^{\max}-\tau_p(k)>0$ and $\Delta\tau^{\min}(k)=1$, if $\tau_p^{\max}-\tau_p(k)\leq 0$. Iteration number L_{WHILE} of the WHILE loop can be estimated as follows: $L_{WHILE} < \max_{k\in\{1,\ldots,K\}}\left\lceil \frac{FT(k)}{\Delta\tau^{\min}(k)}\right\rceil$. It is easy to observe that the complexity of separate steps of the algorithm is as follows: step 5 – $O(N^2)$, step 7 – $O(N)$, step 8 – $O(KN)$, steps 9-11 – $O(1)$. Steps 4-14 are realized in the FOR loop K times, hence the complexity of the algorithm *MSA*.2 is equal $O\left(L_{WHILE}\left(K^2N+KN^2\right)\right)$.

It is possible to improve the value of the objective function (4.19) (and, in consequence, (4.24)) and computational time in the *MSA*.2 algorithm using a preprocessing step (algorithm *MSA*.2.0). In the *MSA*.2.0 algorithm we try to decrease value of objective function (4.19) by decreasing $\forall_{p\in\{1,\ldots,N\}}\Delta\tau_p^{'\max}(k)$ values (for each k-th object), to obtain all non-negative values of $\Delta\tau_p^{'\max}(k)$ (like in the *MSA*.2 algorithm). Let us note that the method of the value of the $a_{s_k}(k)$ selection in the 4th step of the algorithm guarantees, that the value of the cost function will be equal $L(a_{s_k}(k))=0$ (see (4.71)) because of $P_s^{<}(k)=\varnothing$. After running the *MSA.2*.0 algorithm, we start the *MSA.2* algorithm taking into the initialization step the values $\forall_{k\in\{1,\ldots,K\}}\forall_{p\in\{1,\ldots,N\}} x_{k,p}$, $\forall_{k\in\{1,\ldots,K\}} FT(k)$ and $\forall_{k\in\{1,\ldots,K\}}\forall_{p\in\{1,\ldots,N\}}\Delta\tau_p^{'\max}(k)$ obtained from the *MSA*.2.0 algorithm. Computational complexity of the *MSA*.2.0 algorithm can be estimated as follows: external loop FOR realizes K times, number of iteration L_{WHILE} of the WHILE loop for fixed k is bounded by the value L_{WHILE} (like in the *MSA.2* algorithm), step 4 has $O(N)$ complexity, and steps 6-8 – $O(N)$. Hence, the total complexity of the *MSA*.2.0 algorithm is equal $O\left(KL_{WHILE}N\right)$.

Algorithm MSA.2.0

```
Given sets: $I_k$, $T_k$, $IP_k$, $TP_k$ for each $k$=1,…,$K$ ;
Initialize: $\underset{k\in\{1,...,K\}}{\forall}\ \underset{p\in\{1,...,N\}}{\forall} a_p(k):=0$ ; $\underset{k\in\{1,...,K\}}{\forall}\ \underset{p\in\{1,...,N\}}{\forall} x_{k,p}:=0$ ; Exit:=false;
$\underset{k\in\{1,...,K\}}{\forall} FT(k):=\tau^{R_{k^*}}(k^*)-\tau^{R_k}(k)$ ; $\underset{k\in\{1,...,K\}}{\forall}\ \underset{p\in\{1,...,N\}}{\forall} \Delta\tau_p^{'\max}(k):=\tau_p^{\max}-\tau_p(k)$ ;
1.  FOR k=1,…,K DO
2.    WHILE Exit=false DO
3.      IF FT(k)>0  THEN
4.          Find such a minimal value $s_k\in P_1^+(k)$ and maximal value
              $a_{s_k}(k)\in\left(0,\ \min\left\{\max_{p\in\{s_k,...,N\}}\Delta\tau_p^{'\max}(k), FT(k)\right\}\right]$, for which
            condition $\underset{p\in\{s_k,...,N\}}{\forall}\Delta\tau_p^{'\max}(k)-a_{s_k}(k)\geq 0$ is satisfied;
5.          IF $a_{s_k}(k)>0$ THEN
6.             $\underset{p\in\{s_k,...,N\}}{\forall}\Delta\tau_p^{'\max}(k):=\Delta\tau_p^{'\max}(k)-a_{s_k}(k)$ ;
7.             $FT(k):=FT(k)-a_{s_k}(k)$ ;
8.             $x_{k,s_k}:=x_{k,s_k}+a_{s_k}(k)$ ;
9.          ELSE
10.            Exit=true;
11.         END IF;
12.      ELSE
13.        Exit=true;
14.      END IF;
15.    END WHILE;
16.  END FOR;
```

4.2.2.3. Numerical example of using the algorithms

Presented in Fig. 4.5 are examples of using *MSA*.1 and *MSA*.2 algorithms (without using *MSA*.2.0) for *K*=3 objects and *N*=4 checkpoints. It can be observed (Table 4.3) that the value of the criterion function (4.19) before using the *MSA*.2 algorithm is equal to 20 (sum of values in the table excluding the last column) and after using the *MSA*.2 algorithm (Table 4.5) equals 14. Table 4.4 presents initial values of functions $\Delta\tau_p^{'\max}(k)$ and *FT*(*k*) before running algorithm *MSA*.2 (it has been assumed that $T^{\max}=\tau^*$). Table 4.5 contains final values of these functions, after running the *MSA*.2 algorithm. Values of x_{kp} determined by the algorithm are equal zero excluding two values: $x_{3,4}=1$, $x_{2,1}=2$. Let us note that the same solution has been obtained solving the GAMS model in chapter 4.2.1.4. Taking into account values of x_{kp} and formula $\tau_p^{'}(k)=\tau_p(k)+\sum_{i=1}^{p}x_{ki}$ we can obtain modified moments of achieving alignment nodes by all objects (Table 4.6). Taking into account the explanation presented in chapter 4.2.2.1 (after defining algorithm

MSA.1), values of $\tau'_p(k)$ and geometric distances $d_{i^r(k),i^{r+1}(k)}$ between nodes $i^r(k), i^{r+1}(k)$ we can calculate modified velocities $v'^{k}_{i^r(k),i^{r+1}(k)}$ as follows:

$$\underset{k\in\{1,\dots,K\}}{\forall}\ \underset{r\in\{0,\dots,R_k-1\}}{\forall}\ v'^{k}_{i^r(k),i^{r+1}(k)} := \frac{d_{i^r(k),i^{r+1}(k)}}{\tau'^{r+1}(k)-\tau'^{r}(k)}$$

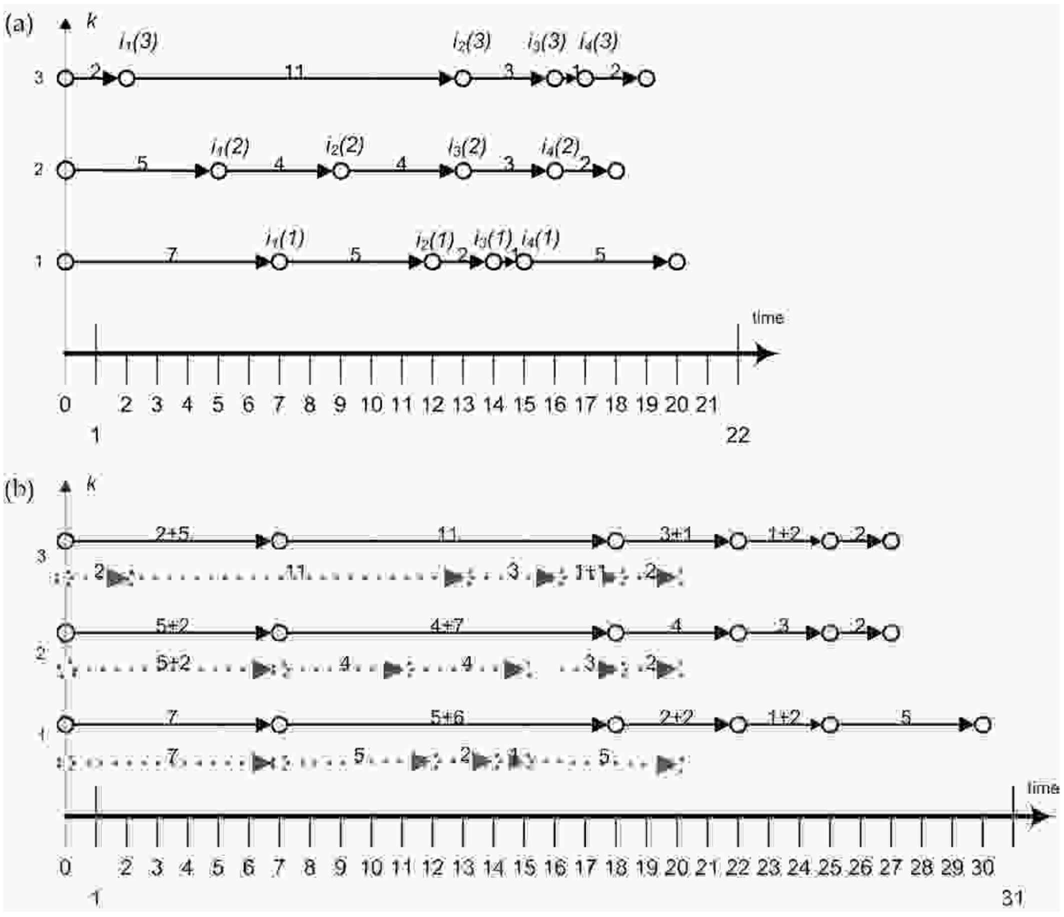


Fig. 4.5. (a) Node-disjoint vector of the shortest paths for *K*=3 objects with achieved times of each *N*=4 alignment nodes for each object; (b) Results of realization of the *MSA*.1 (regular line) and the *MSA*.2 (dashed line) algorithms

In Table 4.7 results of running the *MSA*.2.0 algorithm (before running *MSA*.2) are shown. From the table results that in this preprocessing step we decrease the value of the objective function with 4.

Table 4.8 presents final values of functions $\Delta\tau'^{\max}_p(k)$ and *FT*(*k*) after running the *MSA*.2.0 algorithm.

Table 4.3. Values of functions $\tau_p(k)$ and $\tau^{R_k}(k)$ for example from Fig. 4.5a

k	p				$\tau^{R_k}(k)$
	1	2	3	4	
3	2	13	16	17	19
2	5	9	13	16	18
1	7	12	14	15	20

Table 4.4. Initial values of functions $\Delta\tau_p^{'\max}(k)$ and *FT*(*k*) (before running algorithm *MSA*.2)

k	p				$FT(k)$
	1	2	3	4	
3	5	0	0	0	1
2	2	4	3	1	2
1	0	1	2	2	0

Table 4.5. Final values of functions $\Delta\tau_p^{'\max}(k)$ and *FT*(*k*) (after running algorithm *MSA*.2)

k	p				$FT(k)$
	1	2	3	4	
3	5	0	0	0	0
2	0	2	1	0	0
1	0	1	2	3	0

Table 4.6. Modified moments $\tau_p^{'}(k)$ of achieving checkpoints by all objects (after running algorithm *MSA*.2)

k	p			
	1	2	3	4
3	2	13	16	17+1
2	5+2	9+2	13+2	16+2
1	7	12	14	15

Table 4.7. Results of running algorithm *MSA*.2.0

k	s_k	$a_{s_k}(k)$	$Z(a_{s_k}(k))$	$L(a_{s_k}(k))$
1	0	0	0	0
2	1	1	4	0
3	0	0	0	0

Table 4.8. Final values of functions $\Delta\tau_p^{'\max}(k)$ and *FT*(*k*) (after running algorithm *MSA*.2.0)

k	p				$FT(k)$
	1	2	3	4	
3	5	0	0	0	1
2	1	3	2	0	1
1	0	1	2	2	0

4.2.3. Experimental Analysis of the Algorithms

In Fig. 4.6, Fig. 4.7, Fig. 4.8, Fig. 4.9 the average computational time (on computer with Intel Pentium IV 3GHz processor) in the logarithmic scale [msec] for the *MSA*.2 algorithm, with preprocessing (*MSA*.2.0 before *MSA*.2 algorithm) and without it using different pairs of the *ObjOrder-Strategy* is presented. The size of the problem (4.24)-(4.26) has been set as follows: values of $K \in \{1,\ldots,100\}$ and values of $N \in \{1,\ldots,100\}$ (values of K are divided into a group with a range 10, values of N are grouped into two sets: $1 \le N \le 50$; $51 \le N \le 100$). Over 100 000 randomly generated input data for the problem (4.24)-(4.26) have been examined. To compare obtained results from the *MSA*.2 algorithm, problem (4.24)-(4.26) has been also solved using the GAMS/CONOPT solver (*ObjOrder-Strategy*=-1- -1).

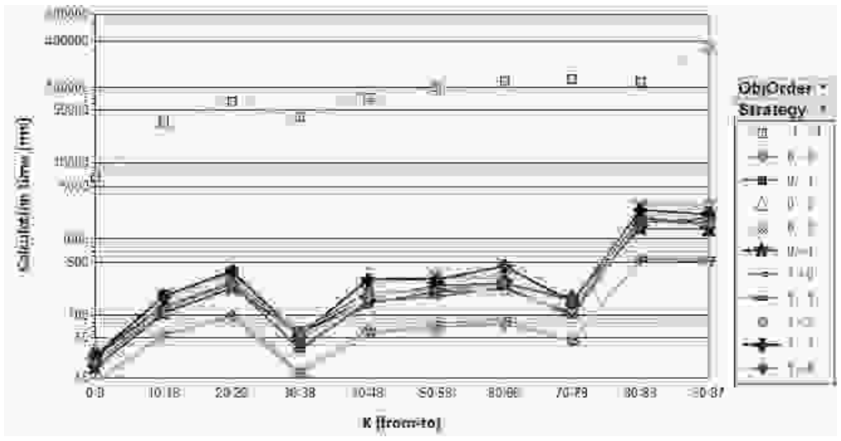


Fig. 4.6. Average computational time in the logarithmic scale [msec] for the *MSA*.2 algorithm, with *Preprocessing*=true (using the *MSA*.2.0 algorithm), *ObjOrder*$\in\{0,1\}$, *Strategy*$\in\{0,\ldots,4\}$, $N\in\{1,\ldots,50\}$; *ObjOrder*=-1 and *Strategy*=-1 deal with solving the nonlinear optimization problem (4.24)-(4.26) using the GAMS/CONOPT solver

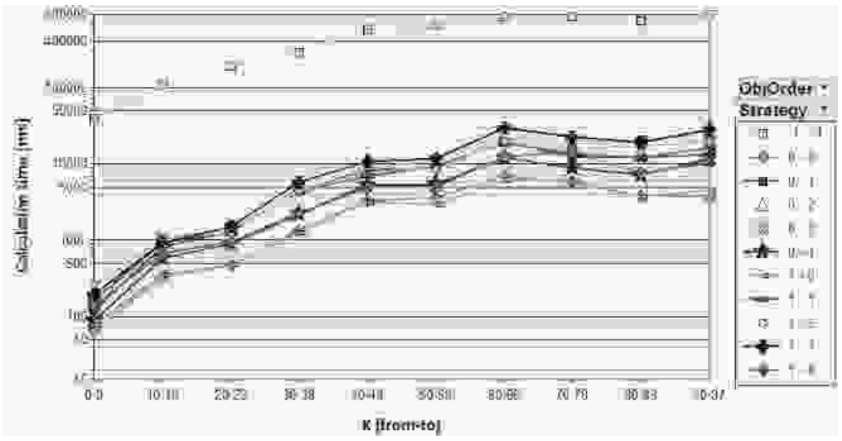


Fig. 4.7. Average computational time in the logarithmic scale [msec] for the *MSA*.2 algorithm, with *Preprocessing*=true (using the *MSA*.2.0 algorithm), *ObjOrder*$\in\{0,1\}$, *Strategy*$\in\{0,\ldots,4\}$, $N\in\{51,\ldots,100\}$; *ObjOrder*=-1 and *Strategy*=-1 deal with solving the nonlinear optimization problem (4.24)-(4.26) using the GAMS/CONOPT solver

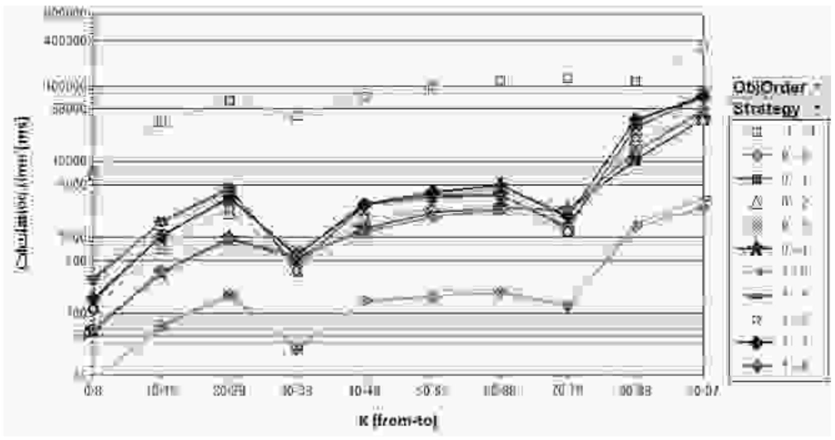


Fig. 4.8. Average computational time in the logarithmic scale [msec] for the *MSA*.2 algorithm, with *Preprocessing*=false (without using the *MSA*.2.0 algorithm), *ObjOrder*∈{0,1}, *Strategy*∈{0,…,4}, *N*∈{1,…,50}; *ObjOrder*=-1 and *Strategy*=-1 deal with solving the nonlinear optimization problem (4.24)-(4.26) using the GAMS/CONOPT solver

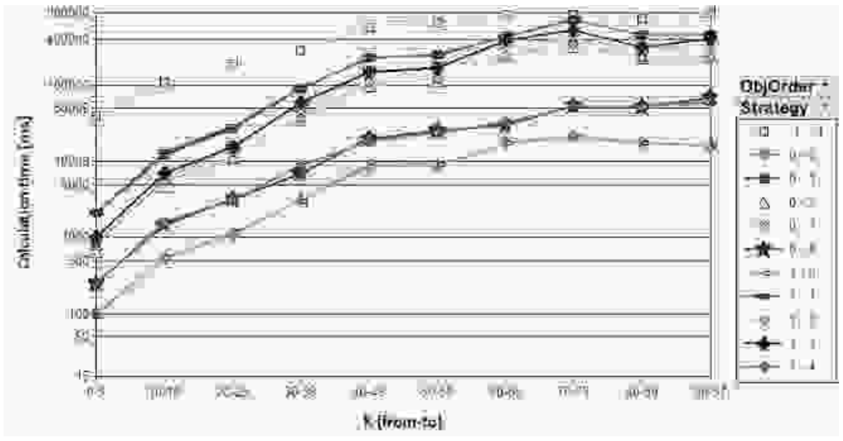


Fig. 4.9. Average computational time in the logarithmic scale [msec] for the *MSA*.2 algorithm, with *Preprocessing*=false (without using the *MSA*.2.0 algorithm), *ObjOrder*∈{0,1}, *Strategy*∈{0,…,4}, *N*∈{51,…,100}; *ObjOrder*=-1 and *Strategy*=-1 deal with solving the nonlinear optimization problem (4.24)-(4.26) using the GAMS/CONOPT solver

It can be observed (comparing Fig. 4.6 and Fig. 4.8 or Fig. 4.7 and Fig. 4.9) that by using the preprocessing step (running algorithm *MSA*.2.0 before *MSA*.2) we can accelerate computational time between a few to twenty times faster than without the preprocessing step. It results from the fact that in the *MSA*.2.0 algorithm we try to decrease the value of the objective function (4.24) by decreasing $\underset{p\in\{1,\ldots,N\}}{\forall} \Delta\tau_p^{'\max}(k)$ values (for each *k*-th object), in order to obtain all nonnegative values of $\Delta\tau_p^{'\max}(k)$ (like in the *MSA*.2 algorithm). Then, the *MSA*.2 algorithm decreases the number of iterations. For all pairs of the *ObjOrder-Strategy* we have obtained faster computational time than when using the GAMS/CONOPT solver. We have obtained the best computational time for the *ObjOrder-Strategy*: 0-0, 1-0 (also for 2-0 and 3-0).

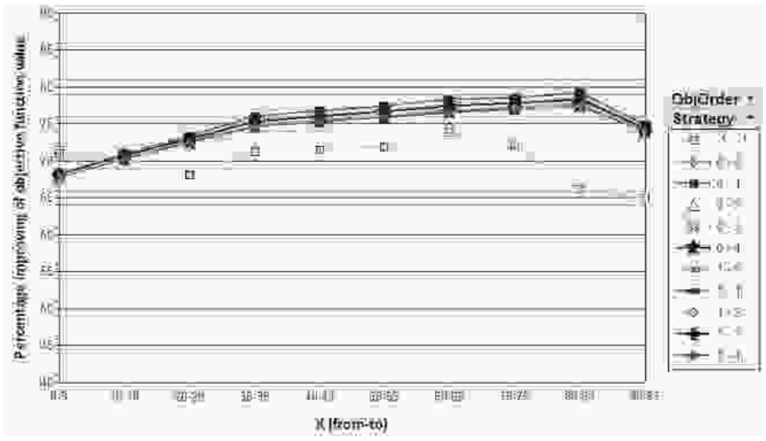


Fig. 4.10. Average percentage improvement of the objective function (4.24) value for the *MSA.2* algorithm, with *Preprocessing*=true (using the *MSA.2.*0 algorithm before *MSA*.2), *ObjOrder*∈{0,1}, *Strategy*∈{0,…,4}, *N*∈{1,…,50}; *ObjOrder*=-1 and *Strategy*=-1 deal with solving the nonlinear optimization problem (4.24)-(4.26) using the GAMS/CONOPT solver

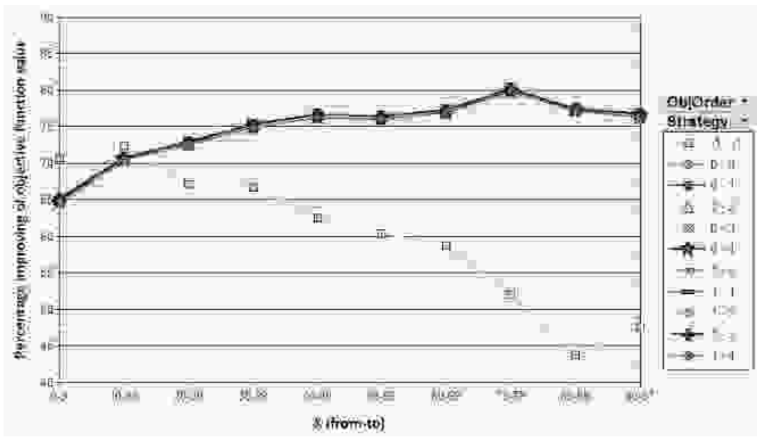


Fig. 4.11. Average percentage improvement of the objective function (4.24) value for the *MSA.2* algorithm, with *Preprocessing*=true (using the *MSA.2.*0 algorithm before *MSA*.2), *ObjOrder*∈{0,1}, *Strategy*∈{0,…,4}, *N*∈{51,…,100}; *ObjOrder*=-1 and *Strategy*=-1 deal with solving the nonlinear optimization problem (4.24)-(4.26) using the GAMS/CONOPT solver

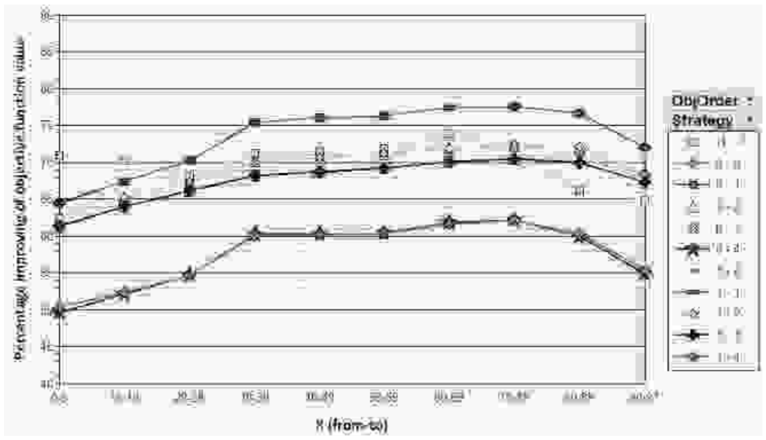


Fig. 4.12. Average percentage improvement of the objective function (4.24) value for the *MSA.2* algorithm, with *Preprocessing*=false (without using the *MSA.2.*0 algorithm before *MSA*.2), *ObjOrder*∈{0,1}, *Strategy*∈{0,…,4}, *N*∈{1,…,50}; *ObjOrder*=-1 and *Strategy*=-1 deal with solving the nonlinear optimization problem (4.24)-(4.26) using the GAMS/CONOPT solver

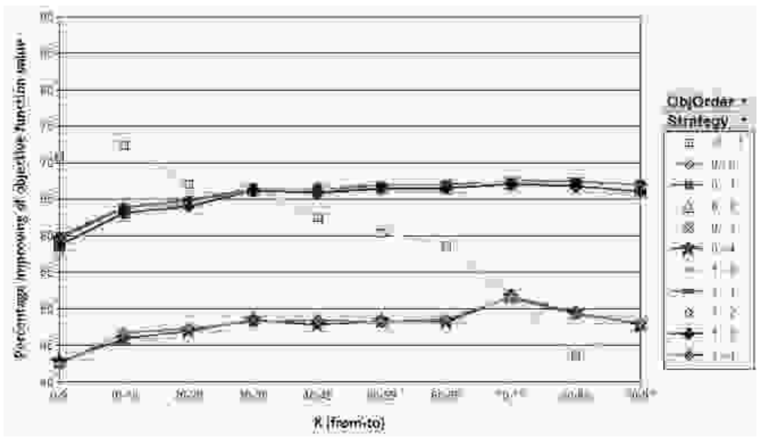


Fig. 4.13. Average percentage improvement of the objective function (4.24) value for the *MSA.2* algorithm, with *Preprocessing*=false (without using the *MSA.2.*0 algorithm before *MSA.*2), *ObjOrder*∈{0,1}, *Strategy*∈{0,…,4}, *N*∈{51,…,100}; *ObjOrder*=-1 and *Strategy*=-1 deal with solving the nonlinear optimization problem (4.24)-(4.26) using the GAMS/CONOPT solver

In Fig. 4.10, Fig. 4.11, Fig. 4.12, Fig. 4.13 the average percentage improvement of the objective function (4.24) value for the *MSA*.2 algorithm, with or without preprocessing (*MSA*.2.0 algorithm) using different pairs of the *ObjOrder-Strategy* is presented (*ObjOrder*∈{0,1}, because for the *ObjOrder*∈{2,3} similar results have been obtained). The percentage improvement value, *PI*, of the objective function value is calculated as follows: $PI = \frac{OBJ_0 - OBJ_1}{OBJ_0} \cdot 100\%$, where OBJ_0, OBJ_1 – values of the objective function (4.24) before and after running the *MSA*.2 algorithm, respectively. For example, $PI = \frac{20-14}{20} \cdot 100\% = 30\%$ for data have been considered in chapter 4.2.2.3. It can be observed that for *K*>20 almost for all pairs of the *ObjOrder-Strategy* in the *MSA*.2 algorithm percentage improvement of the objective function value is better than for using the GAMS/CONOPT solver. This difference grows when the value of *K* grows.

We have obtained the best results using the preprocessing step (Fig. 4.10 and Fig. 4.12) and the following pairs of the *ObjOrder-Strategy*: 0-1, 1-1 (also for 2-1 and 3-1). Percentage improvement of the objective function (4.24) value for the best pairs of the *ObjOrder-Strategy* is equal from 65% to 80%.

4.3. Multicriteria Movement Synchronization Scheduling (*2CMSS* problem)

In chapter 4.2.1.1 two categories of criteria for movement of *K* objects have been defined: C.1 – time category and C.2 – "distance" category. We have taken into consideration the first type of category and we have proposed algorithms for solving one of the problems from this category (chapter 4.2.2).

In this chapter we present one of the formulations of the optimization problem for multicriteria movement synchronization scheduling of K objects taking into account criteria C.1.2 from (4.14) and C.2.1 from (4.15).

4.3.1. Definition of the 2*CMSS* Problem

We consider the following two-criteria optimization problem (taking into account criteria C.1.2 from (4.14) and C.2.1 from (4.15)): in the given graph G (see definition in chapter 4.2.1.1) to find such node-disjoint paths I_k (see (4.1)) visiting specified nodes belonging to IP_k (see (4.5)) for each k-th of K objects and to determine such velocities $v^k_{i^r(k),i^{r+1}(k)}$, $r=\overline{0,R_k-1}$, $k=\overline{1,K}$ that

$$\sum_{k=1}^{K}\tau(I_k)=\sum_{k=1}^{K}\tau^{R_k}(k)\to\min \tag{4.72}$$

$$\sum_{p=1}^{N}\sum_{k=1}^{K}\left(\tau_p^{\max}-\tau_p(k)\right)\to\min \tag{4.73}$$

with constraints: (4.20) and (4.21).
Let us note again that (4.72)=(4.14) and (4.73)=(4.15).
We can formulate this problem as two-criteria optimization problem (nonlinear, discrete-continuous) of determining the K shortest node-disjoint paths via some alignment nodes in the restricted area (*2CMSS* problem) as follows (**A**, **H**, x_{jnk}, out_{ij}, in_{ij}, a_{ink}, h_{ik}, V, M, have been defined in chapter 3.4.2.1, v_{jk} describes the velocity of the k-th object on the j-th arc of graph G and it is equivalent to $v^k_{i^r(k),i^{r+1}(k)}$, d_j is equivalent to $d_{w,w'}$ for the j-th arc represented by a pair of nodes (w,w')):

$$\sum_{j=1}^{A}\sum_{n=1}^{M}\sum_{k=1}^{K}\frac{d_j}{v_{jk}}x_{jnk}\to\min \tag{4.74}$$

$$\sum_{n=1}^{M-1}\sum_{k=1}^{K}\left(\max_{l\in\{1,\dots,K\}}\left(\tau^0(l)+\sum_{b=1}^{A}\frac{d_b}{v_{bl}}\cdot x_{bnl}\right)-\left(\tau^0(k)+\sum_{b=1}^{A}\frac{d_b}{v_{bk}}\cdot x_{bnk}\right)\right)\to\min \tag{4.75}$$

with constraints:

$$\sum_{j=1}^{A}\left(out_{ij}-in_{ij}\right)x_{jnk}=a_{ink},\qquad i=\overline{1,V},\ n=\overline{1,M},\ k=\overline{1,K} \tag{4.76}$$

$$\sum_{j=1}^{A}\sum_{n=1}^{M}\sum_{k=1}^{K}out_{ij}x_{jnk}\le 1,\qquad i=\overline{1,V} \tag{4.77}$$

$$\sum_{j=1}^{A}\sum_{n=1}^{M}\sum_{k=1}^{K}in_{ij}x_{jnk}\le 1,\qquad i=\overline{1,V} \tag{4.78}$$

$$\sum_{j=1}^{A}\sum_{n=1}^{M} out_{ij}x_{jnk} \le h_{ik}, \qquad i=\overline{1,V},\ k=\overline{1,K} \tag{4.79}$$

$$\sum_{j=1}^{A}\sum_{n=1}^{M} in_{ij}x_{jnk} \le h_{ik}, \qquad i=\overline{1,V},\ k=\overline{1,K} \tag{4.80}$$

$$v_{jk} \le v_j^{\max}(k), \qquad j=\overline{1,A},\ k=\overline{1,K} \tag{4.81}$$

$$v_{jk} > 0, \qquad j=\overline{1,V},\ n=\overline{1,M},\ k=\overline{1,K} \tag{4.82}$$

$$x_{jnk} \ge 0, \qquad j=\overline{1,A},\ n=\overline{1,M},\ k=\overline{1,K} \tag{4.83}$$

Let us note that: (4.76)=(3.79), (4.77)=(3.80), (4.78)=(3.81), (4.79)=(3.82), (4.80)=(3.83), (4.83)=(3.84).
We can isolate two subproblems from the 2*CMSS* problem:

- ***NDSP* problem**: function (4.74) (equivalent to (4.72)), constraints (4.76)-(4.80) and (4.83) deal with searching for the *K* node-disjoint paths visiting specified nodes (represented by matrix **A**) and omitting restricted areas (represented by matrix **H**);
- ***MS* problem**: function (4.75) (equivalent to (4.73)), constraints (4.81) (equivalent to (4.20)) and (4.82) (equivalent to (4.21)) deal with searching for such values of velocities on each arc belonging to the path for each object to minimize the total differences between achieving times in all alignment nodes for all objects.

The *NDSP* problem is the same as *NDRP-Sum* (defined in section 3.4.2.1) when we set in *NDRP-Sum*: $d_j := \frac{d_j}{v_{jk}}$.

Interpretation of constraints (4.76)-(4.80) and (4.83) have been described in chapter 3.4.2.1. Constraints (4.81) and (4.82) assure that no stops on each part (arc) of the path for the *k*-th object are permitted (velocity must be greater than zero) and velocity must be no greater than the maximal possible velocity resulting from technical properties of the *k*-th object being moved and topographical properties of the *j*-th arc.

In the presented optimization problem we have *AMK*+*AK* decision variables and *V*(*MK*+*K*+2)+*AK* constraints (excluding (4.82),(4.83)). The problem is very hard to solve (especially for large graphs) even then we can observe that the matrix of the constraint coefficients (built on the basis of the left sides of the constraints (4.76)-(4.80)) is totally unimodular and a_{ink}, h_{ik} (right sides) are integers, hence the constraint (4.83) can be written as $x_{jnk} \ge 0$ (instead of $x_{jnk} \in \{0,1\}$). One of the main difficulty is the problem is nonlinear.

4.3.2. Methods for Solving *2CMSS* Problem

There are several methods to solve multicriteria problems such as *2CMSS*, in generality (Ehrgott, 1997): hierarchization of objective functions (lexicographic solutions), metacriterion functions, compromise solutions, methods with threshold values, etc. Some of them have been described in chapter 3.3.4.

Since the *2CMSS* problem consists of two subproblems: *NDSP* and *MS*, we propose to use the two-stage algorithm to solve the *2CMSS* problem: at first we solved the problem *NDSP* (criteria function (4.74), constraints (4.76)-(4.80) and (4.83)) by replacing v_{jk} by $v_j^{\max}(k)$ in (4.74), $j=\overline{1,A}$, $k=\overline{1,K}$. After solving this problem we obtain node-disjoint shortest paths for all objects; it means that for each (the *j*-th) arc belonging to a path for each (the *k*-th) object we obtain the shortest time-cost arc value equalling $\frac{d_j}{v_j^{\max}(k)}$. Next, we solved the *MS* problem (criteria function (4.75), constraints (4.81) and (4.82)), which is based on making a correction (decreasing) of velocity value $v_{jk} \le v_j^{\max}(k)$ for each of the *j*-th part (arc) of the path, for each *k*-th object to achieve a "parallel movement effect" measured by the value of the function (4.75). This approach corresponds to searching for lexicographic solutions of the *2CMSS* problem. Such a two-stage method for solving presented problems and such a priority order of optimization criteria are quite intuitive: at first we have to find the vector of shortest paths for *K* objects to set optimal paths, under the assumption that we use maximal possible velocities on each arc belonging to the path for each object, and next we try to decrease values of velocities to optimize the second criterion (4.75).

For solving the *NDSP* problem we may use the *SGDP* algorithm described in chapter 3.4.3.1 and for solving the *MS* problem we may use *MSA*.1 or *MSA*.2 algorithms described in chapters 4.2.2.1 and 4.2.2.2.

4.3.3. Numerical Example

In this chapter we present some practical example (corresponding with the problem from Fig. 4.5) of solving the 2*CMSS* problem for the following parameters (see Fig. 4.14a): graph $G=\langle V_G, A_G\rangle$, $V=\overline{\overline{V_G}}=16$, $A=\overline{\overline{A_G}}=120$, $K=3$, $N=4$, $s_1=31$, $s_2=13$, $s_3=1$, $t_1=30$, $t_2=24$, $t_3=12$, $d_j=10$, $j\in\{1,\ldots,A\}$, $i_1(1)=27$, $i_2(1)=22$, $i_3(1)=23$, $i_4(1)=29$, $i_1(2)=15$, $i_2(2)=10$, $i_3(2)=11$, $i_4(2)=17$, $i_1(3)=2$, $i_2(3)=4$, $i_3(3)=5$, $i_4(3)=6$, $v_{31,32}^{\max}(1)=4.29$, $v_{32,26}^{\max}(1)=4.29$, $v_{26,27}^{\max}(1)=4.29$, $v_{27,21}^{\max}(1)=4.0$, $v_{21,22}^{\max}(1)=4.0$, $v_{23,29}^{\max}(1)=10.0$, $v_{29,30}^{\max}(1)=2.0$, $v_{13,14}^{\max}(2)=4.0$, $v_{14,15}^{\max}(2)=4.0$, $v_{15,16}^{\max}(2)=5.0$, $v_{16,10}^{\max}(2)=5.0$, $v_{10,11}^{\max}(2)=2.5$, $v_{11,17}^{\max}(2)=3.33$, $v_{17,18}^{\max}(2)=10.0$, $v_{18,24}^{\max}(2)=10.0$, $v_{1,2}^{\max}(3)=5.0$, $v_{2,3}^{\max}(3)=1.82$, $v_{3,4}^{\max}(3)=1.82$, $v_{4,5}^{\max}(3)=3.33$, $v_{5,6}^{\max}(3)=10.0$, $v_{6,12}^{\max}(3)=5.0$. For all remaining arcs maximal velocities are equal 1.0.

Taking into account the idea of solving the *2CMSS* problem, at first we need to solve the *NDSP* problem using the *SGDP* algorithm and maximal possible velocities. We obtain K=3 node-disjoint shortest paths visiting alignment nodes presented in Fig. 4.14b. To show that the problem corresponds with the problem from Fig. 4.5, let us note that we have obtained the following achieving times of alignment nodes for the k=2 object (see also Table 4.5) using formula (4.7):

for $i_1(2)$=15: $\frac{10}{4.0}+\frac{10}{4.0}=5$, for $i_2(2)$=10 : $5+\frac{10}{5.0}+\frac{10}{5.0}=5+4=9$,

for $i_3(2)$=11 : $9+\frac{10}{2.5}=13$, for $i_4(2)$=17 : $13+\frac{10}{3.33}\approx 16$.

All values of times of achieving alignment nodes are presented in Table 4.5.

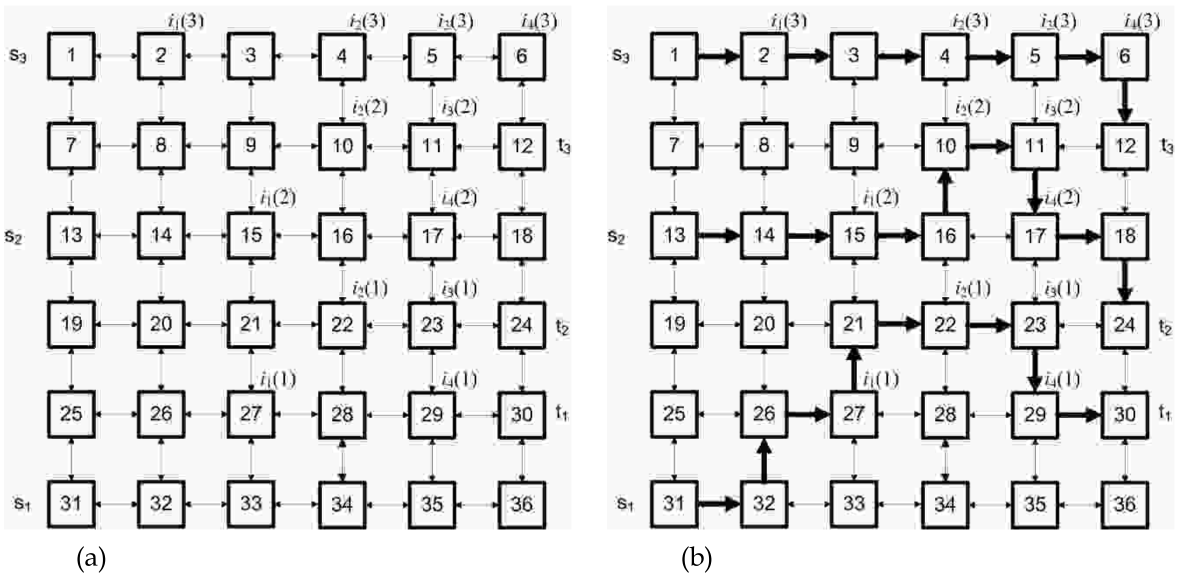


Fig. 4.14. (a) Example of graph with indicated source (s_1,...,s_3), destination (t_1,…,t_3) and alignment nodes ($i_p(k)$, p = 1,..,4, k = 1,...,3) for K=3 objects and N=4 checkpoints for each object; (b) K=3 node-disjoint shortest paths obtained from the *SGDP* algorithm and visiting alignment nodes $i_p(k)$

Next, having paths for the K objects obtained in the previous stage, we can solve the *MS* problem using the *MSA*.2 algorithm. We have obtained modified velocities on arcs belonging to the paths for K objects: $v_{5,6}(3)$ = 5.0, $v_{13,14}(2)$ = $v_{14,15}(2)$ =2.86, the remaining velocities have the same values equal maximal velocities. Modified times of achieving alignment nodes are presented in Table 4.6. For example, for the object k=2 we have obtained following modified times (using formula (4.7)):

for $i_1(2)=15$: $\frac{10}{2.86}+\frac{10}{2.86}\approx 7$, for $i_2(2)=10$: $7+\frac{10}{5.0}+\frac{10}{5.0}=7+4=11$,

for $i_3(2)=11$: $11+\frac{10}{2.5}=15$, for $i_4(2)=17$: $15+\frac{10}{3.33}\approx 18$.

4.4. Summary

In this chapter single- and multi-criteria optimization models and algorithms of movement scheduling for many objects to synchronize their movement (*2CMSS* problem) have been considered. The model consists of two parts: (1) node-disjoint path planning visiting specified nodes for *K* objects with a given vector of intermediate nodes for each one (*NDSP* problem); (2) movement synchronization in intermediate nodes (*MS* problem). The approaches presented in this chapter give possibilities to schedule synchronous movement of many objects and they are used in some simulation-based operational training support systems (Najgebauer *et al.*, 2007b) at the planning stage of action (see also chapter 5.3). It can be shown that they are very fast (in comparison with GAMS/CONOPT (*MSA*.2) or GAMS/CPLEX (*SGDP*) solvers) and it is very important from the point of view of simulator reaction time on user interaction. During movement simulation (movement schedule realization) it is important for movement control and the reaction to deviations from the determined schedule (Tarapata, 2009a). These problems are essential especially in *CGF* or *SAF* systems (Petty, 1995) and they are considered in chapter 5 and chapter 6. Since some of the algorithms being discussed are heuristic (*SGDP, MSA.2*) it seems to be essential to provide necessary and sufficient conditions for obtaining optimal solutions.

It is possible to consider many problems for synchronous movement based on the given approaches: we can modify the problem (4.24)-(4.26) in such a way that in each alignment node neither the delay nor the acceleration of all objects between themselves cannot be greater than the fixed value ΔT, and the criteria function describes the total time of achieving the destination nodes by all objects:

$$\sum_{k=1}^{K}\left(\tau_N(k)+\sum_{i=1}^{N}x_{k,i}\right)\to \min$$

subject to:

$$\sum_{k=1}^{K}\left(\max_{j\in\{1,\dots,K\}}\left(\tau_p(j)+\sum_{i=1}^{p}x_{j,i}\right)-\left(\tau_p(k)+\sum_{i=1}^{p}x_{k,i}\right)\right)\le \Delta T,\quad p=1,\dots,N$$

$$\sum_{p=1}^{N}x_{k,p}\le FT(k),\qquad k=1,\dots,K$$

$$x_{k,p}\ge 0,\quad k=1,\dots,K;\quad p=1,\dots,N$$

Presented suggestions may contribute to further works.

The chapter is organized as follows. Chapters 5.2 and 5.3 (based on the papers (Tarapata 2007b; 2007e; 2008b; 2008c; 2010b)) contain description of automatization methods of the main battlefield processes (attack, defence and march) in simulation systems such as *CGF*. In these chapters, a decision automata, which is a component of the simulation system for military training, is described as an example. In chapter 5.4 (based on the papers (Tarapata 2000b; 2000f; 2003a; 2005a; 2010b)) we present methods for movement simulation of individual and group objects based on the MODSIM simulation language. Presented in chapter 5.5 are some conclusions concerning problems and proposition of their solution in automatization of decision processes in conflict situations.

5.2. Identification of Decision Situations

5.2.1. Description and Definition of the Problem

The typical military decision planning process contains the following steps (see Fig. 5.1):

- estimation of power of own and opposite forces, terrain, and other factors, which may influence on a task realization,
- identification of a decision situation,
- determination of decision variants (*Course of Actions*, *CoA*),
- variants (*CoA*) evaluation (verification),
- recommendation of the best variant (*CoA*) of the above-stated points, which satisfy the proposed criteria.

The most important step of the decision planning process is an identification of the decision situation problem: this problem is that we must find the most similar battlefield situation (from earlier defined or ensuing situations, e.g. in knowledge base of battlefield situations, see Fig. 1.1) to the current one. Afterwards, the decision situation being identified is a basis for choosing *CoA*, because with each decision situation a few typical *CoA* frames (templates) are connected. The decision situation is classified according to the following factors: own task, expected actions of opposite forces, environmental conditions – terrain, weather, the time of the day and season of the year, current state of own and opposite forces in the sense of personnel and weapon systems.

We define space of decision situations as follows:

$$DSS = \{SD:\ SD = (SD_r)_{r=1,\ldots,8}\} \tag{5.1}$$

Vector *SD* represents the decision situation, which is described by the following eight elements: SD_1 – command level of opposite forces, SD_2 – type of task of opposite forces (e.g. attack, defence), SD_3 – command level of own forces,

SD_4 – type of task of own forces (e.g. attack, defence), SD_5 – net of squares as a model of activities (terrain) area $SD_5 = \left[SD^5_{ij}\right]_{\substack{i=1,..,SD_7 \\ j=1,..,SD_8}}$, $SD^5_{ij} = (SD^{5,k}_{ij})_{k=1,..,8}$.

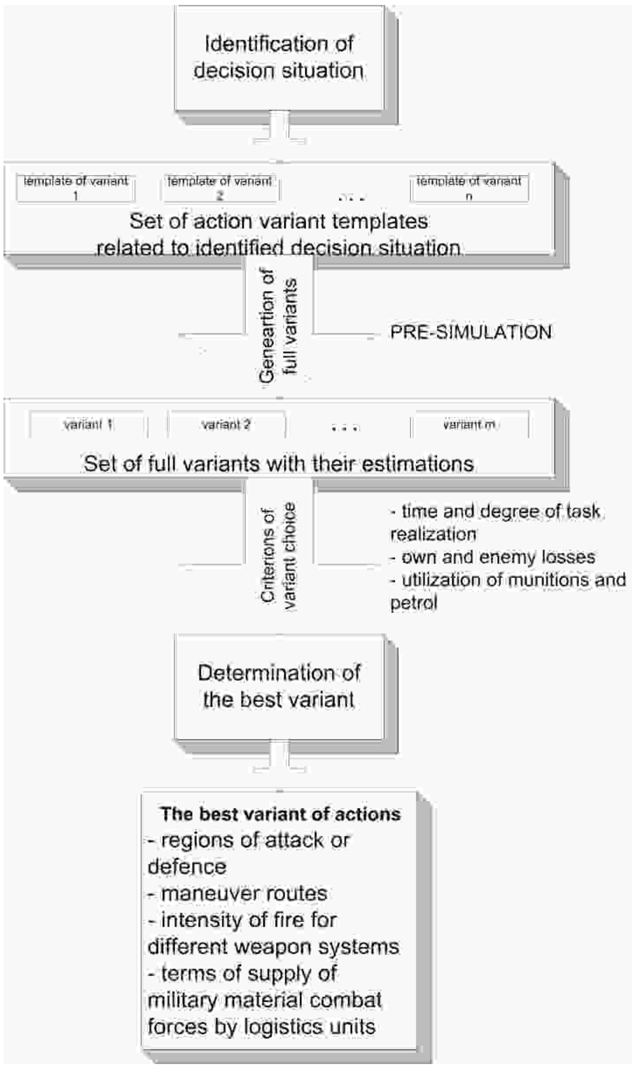

Identification of decision situation
template of variant 1
template of variant 2
...
template of variant n
Set of action variant templates related to identified decision situation
Geneartion of full variants
PRE-SIMULATION
variant 1
variant 2
...
variant m
Set of full variants with their estimations
Criterions of variant choice
- time and degree of task realization
- own and enemy losses
- utilization of munitions and petrol
Determination of the best variant
The best variant of actions
- regions of attack or defence
- maneuver routes
- intensity of fire for different weapon systems
- terms of supply of military material combat forces by logistics units

Fig. 5.1. Algorithm for selecting the best variant of action (Antkiewicz *et al.*, 2005)

For the terrain square with the indices (*i,j*) each of the elements denotes: $SD^{5,1}_{ij}$ – the degree of terrain passability, $SD^{5,2}_{ij}$ – the degree of forest covering, $SD^{5,3}_{ij}$ – the degree of water covering, $SD^{5,4}_{ij}$ – the degree of terrain undulating, $SD^{5,5}_{ij}$ – armoured power (potential) of opposite units, $SD^{5,6}_{ij}$ – infantry power (potential) of opposite units, $SD^{5,7}_{ij}$ – artillery power (potential) of opposite units, $SD^{5,8}_{ij}$ – coordinates of the square, SD_6 – the description of own forces: $SD_6 = \left(SD^6_i\right)_{i=1,..,4}$, SD^6_1 – total armoured power (potential), SD^6_2 – total infantry power (potential),

SD_3^6 – total artillery power (potential), SD_4^6 – total air fire support (antiaircraft) power (potential); SD_7 – the width of activities (interest) in an area (number of squares), SD_8 – the depth of activities (interest) in an area (number of squares).

The set of decision situations patterns is given: $PDSS = \{PS : PS \in DSS\}$. For the current decision situation *CS*, we have to find the most similar situation *PS* from the set of patterns. In chapters 5.2.2 and 5.2.3 we present more formal definitions of "situations similarity".

We have determined the subset of decision situation patterns $PDSS_{CS}$, which are generally similar to the current situation *CS*, considering such elements like: task type, command level of own and opposite units and own units' potential:

$$PDSS_{CS} = \left\{ PS = (PS_i)_{i=1,..,6} \in PDSS : PS_i = CS_i,\, i = 1,..,4,\, dist_{potwl}(CS, PS) \le \Delta Pot \right\} \quad (5.2)$$

where:

$$dist_{potwl}(CS, PS) = \max\left\{ \left| CS_k^6 - PS_k^6 \right|, k = 1,..4 \right\} \quad (5.3)$$

and ΔPot – the maximum difference of the potential of own forces (calibration parameter).

5.2.2. Distance Vector Approach

Here, we present the distance vector approach for solving the problem defined in chapter 5.2.1. We formulated and solved the multicriteria optimization problem (5.4), which allow us to determine the most matched pattern situation (*PS*) to the current one (*CS*) from the point of view of terrain and military power characteristics (Najgebauer *et al.*, 2007b):

$$Z = \left(PDSS_{CS}, F_{CS}, R_D \right) \quad (5.4)$$

where:

$$F_{CS} : PDSS_{CS} \to R^2 \quad (5.5)$$

$$F_{CS}(PS) = \left(dist_{ter}(CS,PS),\, dist_{pot}(CS,PS) \right) \quad (5.6)$$

$$dist_{ter}(CS,PS) = \sum_{k=1}^{4} \lambda_k \cdot \left(\sum_{i=1}^{I} \sum_{j=1}^{J} \left(CS_{ij}^{5,k} - PS_{ij}^{5,k} \right)^p \right)^{\frac{1}{p}} \quad (5.7)$$

$$\sum_{k=1}^{4} \lambda_k = 1,\, \lambda_k > 0,\, k = 1,..,4 \quad (5.8)$$

$$dist_{pot}(CS,PS) = \sum_{k=5}^{7} \mu_k \cdot \left(\sum_{i=1}^{I} \sum_{j=1}^{J} \left(CS_{ij}^{5,k} - PS_{ij}^{5,k} \right)^p \right)^{\frac{1}{p}} \quad (5.9)$$

$$\sum_{k=5}^{7} \mu_k = 1,\, \mu_k > 0,\, k = 5,..,7 \quad (5.10)$$

$$I = \min\{CS_7, PS_7\},\; J = \min\{CS_8, PS_8\} \quad (5.11)$$

$$R_D = \left\{ \begin{array}{l} (Y,Z) \in PDSS_{CS} \times PDSS_{CS} : \\ dist_{ter}(CS,Y) \le dist_{ter}(CS,Z) \wedge \\ dist_{pot}(CS,Y) \le dist_{pot}(CS,Z) \end{array} \right\} \tag{5.12}$$

Parameters μ_k and λ_k describes the weights for components calculating the value of functions $dist_{ter}$ and $dist_{pot}$. The domination relation defined in (5.12) allows us to choose such a *PS* from $PDSS_{CS}$, which has the best value of $dist_{ter}$ and $dist_{pot}$, that is the most similar to *CS* (non-dominated *PS* from the R_D point of view). The idea of the identification of the decision situation and *CoA* selection is presented in Fig. 5.2. Application of this method is presented in chapter 6.2 and in (Antkiewcz *et al.*, 2011b).

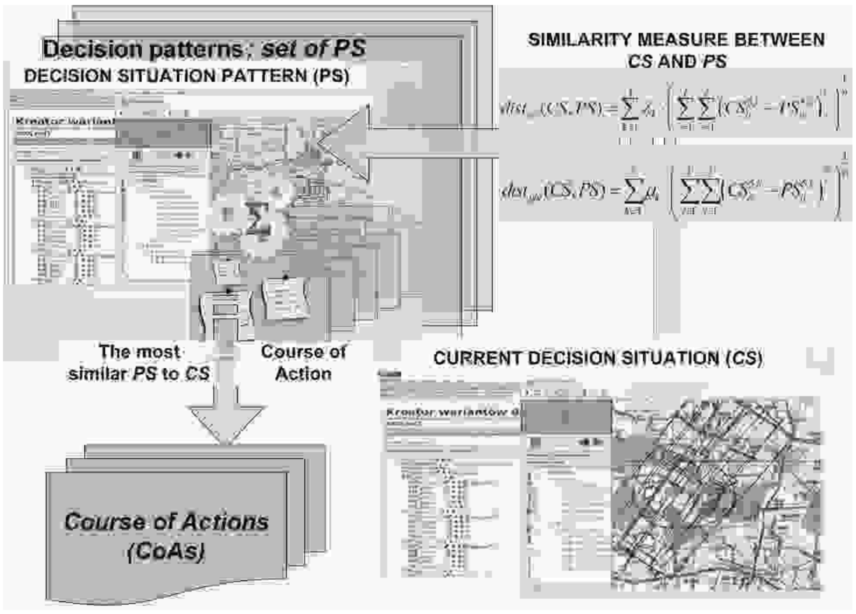

Decision patterns: set of PS
DECISION SITUATION PATTERN (PS)
SIMILARITY MEASURE BETWEEN CS AND PS
The most similar PS to CS
Course of Action
CURRENT DECISION SITUATION (CS)
Course of Actions (CoAs)

Fig. 5.2. The idea of identification of the decision situation and *CoA* selection (Antkiewicz *et al.*, 2011b)

5.2.3. Multicriteria Weighted Graphs Similarity (*MWGSP*) Approach

In this chapter concept of multicriteria weighted graphs similarity and its application for pattern matching of decision situations is considered. The approach extends known pattern recognition approaches based on graph similarity with two features: (1) the similarity is calculated as structural and non-structural (quantitative) in a weighted graph, (2) choice of the most similar graph to graph representing pattern is based on a multicriteria decision. Application of the presented approach for pattern recognition of decision situations has been described in (Tarapata, 2007b; 2008b) and in chapter 5.2.3.5.

5.2.3.1. Structural objects similarity – a short overview

Object similarity is an important issue in applications such as pattern recognition. With given a database of known objects and a pattern, the task is to retrieve one or several objects from the database that are similar to the pattern.

If graphs are used for object representation this problem turns into determining the similarity of graphs, which is generally referred to as graph matching. Standard concepts in graph matching include (Farin *et al.*, 2003; Kriegel & Schonauer, 2003): graph isomorphism, subgraph isomorphism, graph homomorphism, maximum common subgraph, error-tolerant graph matching using graph edit distance (Bunke, 1997), graph's vertices similarity, histograms of the degree sequence of graphs. A large number of applications of graph matching have been described in the literature (Bunke, 2000; Kriegel & Schonauer, 2003; Robinson, 2004). One of the earliest applications was in the field of chemical structure analysis. More recently, graph matching has been applied to case-based reasoning, machine learning planning, machine vision, semantic networks, social networks, conceptual graph, monitoring of computer networks, synonym extraction and web searching (Bunke, 2000; Blondel *et al.*, 2004; Champin & Solnon, 2003; Kleinberg, 1999; Kriegel & Schonauer, 2003; Melnik *et al.*, 2002; Robinson, 2004; Senellart & Blondel, 2003; Tarapata & Kasprzyk, 2009c; 2010e; Tarapata *et al.*, 2010d). They include recognition of graphical symbols, character recognition, shape analysis, terrorist network analysis, three-dimensional object recognition, image and video indexing and others. It seems that structural similarity is not sufficient for similarity description between various objects. The arc in the graph gives only binary information concerning connection between two nodes. And what about, for example, the connection strength, connection probability or other characteristics? Thus, the weighted graph matching problem is defined, but in the literature it is relatively rarely considered (Almohamad & Duffuaa, 1993; Champin & Solnon, 2003; Tarapata, 2007b; Umeyama, 1988) and it is most often regarded as a special case of graph edit distance, which is a very time-complex measure (Bunke, 2004; Kriegel & Schonauer, 2003). Therefore, we define a multicriteria weighted graph similarity decision problem (*MWGSP*) and we show how to use it for pattern recognition (matching) of decision situations (*PRDS*) in the decision automata, which replaces commanders in simulators for military trainings (Najgebauer *et al.*, 2007b).

5.2.3.2. Definitions of structural and quantitative similarity measures between weighted graphs

Let us define weighted graph *WG* as follows:

$$WG = \left\langle G, \{f_i(n)\}_{\substack{i\in\{1,\ldots,LF\}\\ n\in N_G}}, \{h_j(a)\}_{\substack{j\in\{1,\ldots,LH\}\\ a\in A_G}} \right\rangle \tag{5.13}$$

where: *G* – Berge's graph, $G=\langle N_G, A_G\rangle$, *N*$N_G$, A_G – sets of graph's nodes and arcs, $A_G \subset \{\langle n,n'\rangle : n,n' \in N_G\}$, $f_i : N_G \to R^n$ – the *i*-th function described on the graph's nodes, $i=1,...LF$, (*LF* – number of node's functions); $h_j : A_G \to R^n$ – the *j*-th function described on the graph's arcs, $j=1,...LH$ (*LH* – number of arc functions).

Let two weighted graphs G_A and G_B be given. We propose to calculate two types of similarities of the G_A and G_B: structural and non-structural (quantitative). To calculate structural similarity between G_A and G_B it is proposed to use the approach defined in (Blondel *et al.*, 2004). Let *A* and *B* be the transition matrices of G_A and G_B. We calculate the following sequence of matrices:

$$Z_{k+1}=\frac{BZ_kA^T+A^TZ_kB}{\left\|BZ_kA^T+A^TZ_kB\right\|_F},\quad k\geq 0 \tag{5.14}$$

where Z_0=**1** (matrix with all elements equal 1); x^T – matrix *x* transposition; $\|x\|_F$ – Frobenius (Euclidian) norm for matrix *x*, $\|x\|_F=\sqrt{\sum_{i=1}^{n_B}\sum_{j=1}^{n_A}x_{ij}^2}$, n_B – number of matrix rows (number of nodes of G_B), n_A – number of matrix columns (number of nodes of G_A). Element z_{ij} of the matrix *Z* describes the similarity score between the *i*-th node of G_B and the *j*-th node of G_A. The essence of the similarity of the graph nodes is the fact that two graph nodes are similar, if their neighbouring nodes are similar. The greater value of z_{ij} the greater the similarity between the *i*-th node of G_B and the *j*-th node of G_A. We obtain structural similarity matrix $S(G_A,G_B)$ between nodes of graphs G_A and G_B as follows:

$$S(G_A,G_B)=[s_{ij}]_{n_B\times n_A}=\lim_{k\to+\infty}Z_{2k} \tag{5.15}$$

Some computation aspects of calculation $S(G_A,G_B)$ have been presented in (Blondel *et al.*, 2004). We can write (5.14) more explicitly by using the matrix-to-vector operator that develops a matrix into a vector by taking its columns one by one. Therefore, we can write the equality (5.14) as follows:

$$z_{k+1}=\frac{(A\otimes B+A^T\otimes B^T)z_k}{\left\|(A\otimes B+A^T\otimes B^T)z_k\right\|_F} \tag{5.16}$$

where "⊗" denotes the Kronecker product (also denoted tensorial, direct or categorial product). Unfortunately, iteration z_{k+1} does not always converge. Authors of the work (Melnik *et al.*, 2002) showed that if we change the formula (5.16) for $z_{k+1}=\frac{(A\otimes B+A^T\otimes B^T)z_k+b}{\left\|(A\otimes B+A^T\otimes B^T)z_k+b\right\|_F}$, then formula (5.16) converges for *b*>0. Having matrix $S(G_A,G_B)$, we can formulate and solve an optimal assignment problem (using e.g. the Hungarian algorithm) to find the best allocation matrix $X = [x_{ij}]_{n_B \times n_A}$ of nodes from graph describing *GA, GB*:

$$d_S(G_A, G_B) = \sum_{i=1}^{n_B} \sum_{j=1}^{n_A} s_{ij} \cdot x_{ij} \to \max \tag{5.17}$$

with constraints:

$$\sum_{i=1}^{n_B} x_{ij} \le 1, \quad j = \overline{1, n_A} \tag{5.18}$$

$$\sum_{j=1}^{n_A} x_{ij} \le 1, \quad i = \overline{1, n_B} \tag{5.19}$$

$$\underset{i \in \{1,\ldots,n_B\}}{\forall} \; \underset{j \in \{1,\ldots,n_A\}}{\forall} x_{ij} \in \{0,1\} \tag{5.20}$$

The $d_S(G_A,G_B)$ describes the value of *structural similarity measure* of G_A and G_B (Fig. 5.3). Let us note that we can easily adopt centrality measures from social networks to use them or their combinations instead s_{ij} (Bartosiak *et al.*, 2011).

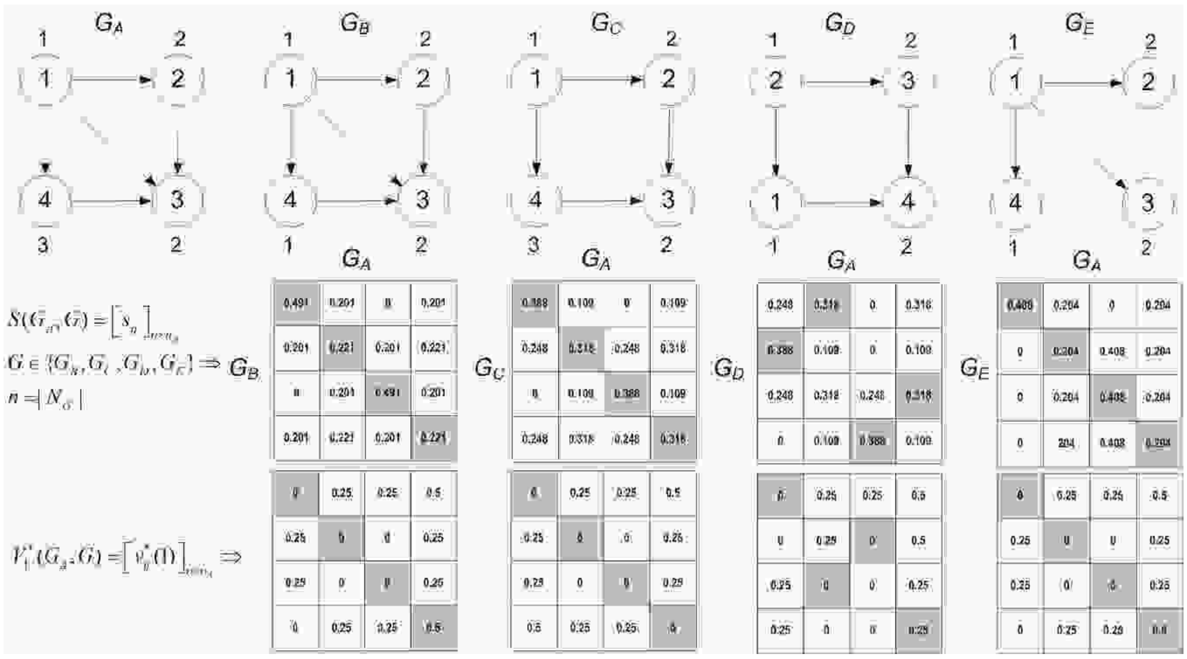
To calculate non-structural (quantitative) similarity between G_A and G_B we should consider the similarity between values of node and arc functions (*nodes and arcs quantitative similarity*), (Tarapata, 2007b). To compute quantitative similarity of nodes we propose to create a vector $\boldsymbol{v}(G_A, G_B) = \langle V_1, \ldots, V_{LF} \rangle$ of matrices, where $V_k = \left[v_{ij}(k) \right]_{n_B \times n_A}$, *k*=1,…,*LF*, describing similarity matrix between nodes of G_A and G_B from the point of view of the *k*-th node's function ($f_k^A : N_{G_A} \to R^n$ for G_A and $f_k^B : N_{G_B} \to R^n$ for G_B) and $v_{ij}(k) = \left\| f_k^B(i) - f_k^A(j) \right\|$ describes the "distance" between the *i*-th node of G_B and the *j*-th node of G_A from the point of view of f_k^B and f_k^A, respectively. We can apply a norm with parameter $p \ge 1$ as distance measure:

$$\left\| f_k^B(i) - f_k^A(j) \right\| = \left\| f_k^B(i) - f_k^A(j) \right\|_p = \left(\sum_{r=1}^{n} \left| f_{k,r}^B(i) - f_{k,r}^A(j) \right|^p \right)^{1/p} \tag{5.21}$$

where $f_{k,r}^A(\cdot)$, $f_{k,r}^B(\cdot)$ describe the *r*-th component of the vector being the value of f_k^A and f_k^B, respectively.

Next, we compute for each *k*=1,…,*LF* normalized matrix $V_k^* = \left[v_{ij}^*(k) \right]_{n_B \times n_A}$, where $v_{ij}^*(k) = v_{ij}(k) \big/ \left\| V_k \right\|_F$. This procedure guarantees that each $v_{ij}^*(k) \in [0,1]$. Finally, we compute the total quantitative similarity between the *i*-th node of G_B and the *j*-th node of G_A as follows:

$$\bar{v}_{ij} = \sum_{k=1}^{LF} \lambda_k \cdot v_{ij}^*(k), \quad \sum_{k=1}^{LF} \lambda_k = 1, \quad \underset{k=1,\ldots,LF}{\forall} \lambda_k \in [0,1] \tag{5.22}$$


G_A
G_B
G_C
G_D
G_E
$S(G_A,G)=[s_{ij}]_{n\times n_A}$
$G\in\{G_B,G_C,G_D,G_E\}\Rightarrow$
$n=|N_G|$
$V_1^*(G_A,G)=[v_{ij}^*(1)]_{n\times n_A}\Rightarrow$

$S(G_A,G_B)$:

0.491	0.201	0	0.201
0.201	0.221	0.201	0.221
0	0.201	0.491	0.201
0.201	0.221	0.201	0.221

$S(G_A,G_C)$:

0.388	0.109	0	0.109
0.248	0.318	0.248	0.318
0	0.109	0.388	0.109
0.248	0.318	0.248	0.318

$S(G_A,G_D)$:

0.248	0.318	0	0.318
0.388	0.109	0	0.109
0.248	0.318	0.248	0.318
0	0.109	0.388	0.109

$S(G_A,G_E)$:

0.408	0.204	0	0.204
0	0.204	0.408	0.204
0	0.204	0.408	0.204
0	204	0.408	0.204

$V_1^*(G_A,G_B)$:

0	0.25	0.25	0.5
0.25	0	0	0.25
0.25	0	0	0.25
0	0.25	0.25	0.5

$V_1^*(G_A,G_C)$:

0	0.25	0.25	0.5
0.25	0	0	0.25
0.25	0	0	0.25
0.5	0.25	0.25	0

$V_1^*(G_A,G_D)$:

0	0.25	0.25	0.5
0	0.25	0	0.5
0.25	0	0	0.25
0.25	0	0	0.25

$V_1^*(G_A,G_E)$:

0	0.25	0.25	0.5
0.25	0	0	0.25
0.25	0	0	0.25
0	0.25	0.25	0.0

Fig. 5.3. Examples of weighted graphs with a single function described on the nodes (set of functions described on the arcs is empty) and their structural ($S(G_A,G)$) and quantitative ($V_1^*(G_A,G)$) similarity matrices. Dark filled cells describe ones, which create optimal assignment of the nodes of G_A to nodes of $G\in\{G_B, G_C, G_D, G_E\}$

The $d_{QN}(G_A,G_B)$ *nodes quantitative similarity measure* of G_A and G_B we compute by solving the assignment problem (5.17)-(5.19) substituting $-\bar{v}_{ij}$ for s_{ij} (because of that the smaller value of $\bar{v}_{ij}$ the better) and $d_{QN}(G_A,G_B)$ for $d_S(G_A,G_B)$ in (5.17).

An example of calculations similarity matrices between nodes of graphs and similarity measures d_S and d_{QN} between graphs are presented in Fig. 5.3 and in Table 5.1. Let us note that the best structural matched graph for G_A is G_B ($d_S(G_A,G_B)$=1.423 is the maximal value among values of this measure for other graphs) but the best quantitative matched graph for G_A is G_C ($d_{QN}(G_A,G_C)$=0 is a minimal value among of values of this measure for other graphs). The question is: which graph is the most similar to G_A : G_B or G_C? A method for solving the problem and to answer the question is presented in chapter 5.2.3.4: we have to apply a multicriteria choice of the best matched graph to G_A.

We can obtain *arcs quantitative similarity measure* $d_{QA}(G_A,G_B)$ by analogy to $d_{QN}(G_A,G_B)$: we build a vector $\boldsymbol{e}(G_A,G_B)=\langle E_1,\dots,E_{LH}\rangle$ of matrices, where $E_k=[e_{ij}(k)]_{m_B\times m_A}$, k=1,…,LH (m_A, m_B – number of arcs in G_A and G_B) describing the similarity matrix between arcs of G_A and G_B from the point of view of the k-th arc function ($h_k^A: A_{G_A}\to R^n$ for G_A and $h_k^B: A_{G_B}\to R^n$ for G_B), $e_{ij}(k)=\left\|h_k^B(i)-h_k^A(j)\right\|_p$, next $e_{ij}^*(k)=e_{ij}(k)/\left\|E_k\right\|_F$ and $\bar{e}_{ij}=\sum_{k=1}^{LH}\mu_k\cdot e_{ij}^*(k)$, $\sum_{k=1}^{LH}\mu_k=1$, $\underset{k=1,\dots,LH}{\forall}\mu_k\ge 0$. Substituting in

(5.17) $-\bar{e}_{ij}$ for s_{ij}, $d_{QA}(G_A,G_B)$ for $d_S(G_A,G_B)$ and solving (5.17)-(5.19) we obtain $d_{QA}(G_A,G_B)$.

Table 5.1. Values of similarity measures between G_A and each of the four graphs from Fig. 5.3

Graph G	$d_S(G_A,G)$	$d_{QN}(G_A,G)$	$0.5d_S(G_A,G) - 0.5d_{QN}(G_A,G)$
G_B	**1.423**	0.5	0.462
G_C	1.412	**0**	**0.706**
G_D	1.412	0.25	0.456
G_E	1.225	0.5	0.362

Let us note that it is possible to determine a single quantitative similarity measure for G_A and G_B. To this end, we use transformation of graph $G=\langle N,A\rangle$ into a temporary graph $G^*=\langle N^*,A^*\rangle$ as follows: $N^*=N\cup A$, $A^*\subset N^*\times N^*$ and

$$\underset{v\in N, a\in A}{\forall}\left(\underset{x\in N}{\exists}(v,x)=a\Rightarrow(v,a)\in A^*\right)\vee\left(\underset{x\in N}{\exists}(x,v)=a\Rightarrow(a,v)\in A^*\right) \quad (5.23)$$

If G was a weighted graph then in G^* we attribute the arc and node functions from G to appropriate nodes of G^* (that is to nodes and arcs from G). Using this procedure for G_A and G_B we obtain G_A^* and G_B^*. Next, for G_A^* and G_B^* we can calculate a quantitative similarity measure $d_{QN}(G_A^*,G_B^*)$ of nodes. Example of constructing G^* from G is presented in Fig. 5.4.

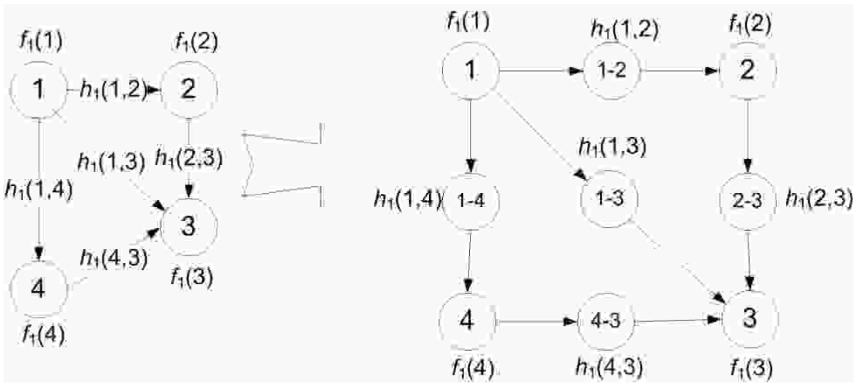


Fig. 5.4. Transformation of G (left-hand side) into G^* (right-hand side)

5.2.3.3. Epsilon-similarity of weighted graphs

At this moment, we propose another view on the quantitative similarity between weighted graphs (Tarapata, 2007b).

Definition 5.1

Let us two weighted graphs:

$$WG_A = \left\langle G_A = \left\langle V_A, E_A \right\rangle, \{f^A(n)\}_{n\in V_A}, \varnothing \right\rangle \text{ and } WG_B = \left\langle G_B = \left\langle V_B, E_B \right\rangle, \{f^B(n)\}_{n\in V_B}, \varnothing \right\rangle$$

be given and $f^A : V_A \to R$, $f^B : V_B \to R$.

We say that node $x \in V_A$ *is (f,ε)-similar to node* $y \in V_B$, $\varepsilon \geq 0$, *if*

$f^A(x) \in [(1-\varepsilon)\cdot f^B(y);\ (1+\varepsilon)\cdot f^B(y)]$ *or* $f^B(y) \in [(1-\varepsilon)\cdot f^A(x);\ (1+\varepsilon)\cdot f^A(x)]$.

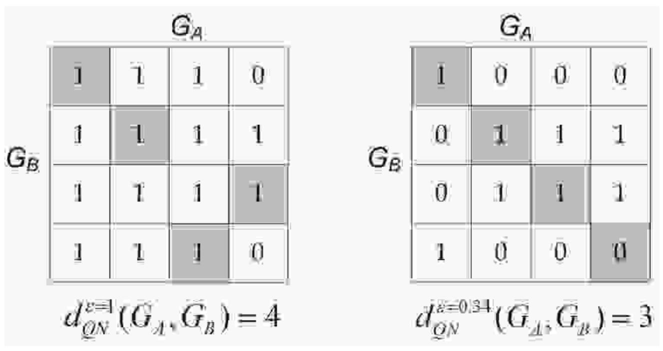
We can use the definition of (*f*,*ε*)-similarity of nodes to construct (*f*,*ε*)-similarity measure between graphs *G_A* and *G_B*. To this end we define the binary similarity matrix between nodes of G_A and G_B as follows: $V^b(\varepsilon) = [v^b_{ij}(\varepsilon)]_{n_B \times n_A}$ and $v^b_{ij}(\varepsilon) = 1$ if $f^A(j) \in [(1-\varepsilon)\cdot f^B(i);\ (1+\varepsilon)\cdot f^B(i)]$ or

$f^B(i) \in [(1-\varepsilon)\cdot f^A(j);\ (1+\varepsilon)\cdot f^A(j)]$, $i \in V_B$, $j \in V_A$ and $v^b_{ij}(\varepsilon) = 0$ otherwise. Next we compute $v^{b*}_{ij}(\varepsilon) = v^b_{ij}(\varepsilon) / \left\| V^b(\varepsilon) \right\|_F$, and compute $d^{\varepsilon}_{QN}(G_A, G_B)$ solving the assignment problem (5.17)-(5.19) by substituting $v^{b*}_{ij}(\varepsilon)$ for s_{ij} and $d^{\varepsilon}_{QN}(G_A, G_B)$ for $d_S(G_A,G_B)$ in (5.17). This idea may be easily extended on a set of node functions.

The idea of the (*f*,*ε*)-similarity is presented in Fig. 5.5. Weighted graphs G_A and G_B with a single function described on the nodes are defined in Fig. 5.3. We obtain, for example:

$v^b_{1,3}(\varepsilon = 1) = 1$ because $f^A(3) = 1 \in [(1-1)\cdot f^B(1);\ (1+1)\cdot f^B(1)]$ that is $f^A(3) = 1 \in [0;\ 2\cdot 2]$;

$v^b_{3,4}(\varepsilon = 0.34) = 1$ because $f^B(3) = 2 \in [(1-0.34)\cdot f^A(4) = 3;\ (1+0.34)\cdot f^A(4) = 3]$ that is $f^B(3) = 2 \in [0.66\cdot 3;\ 1.34\cdot 3]$.



G_A

G_B				
	1	1	1	0
	1	1	1	1
	1	1	1	1
	1	1	1	0

$d^{\varepsilon=1}_{QN}(G_A, G_B) = 4$

G_A

G_B				
	1	0	0	0
	0	1	1	1
	0	1	1	1
	1	0	0	0

$d^{\varepsilon=0.34}_{QN}(G_A, G_B) = 3$

Fig. 5.5. The idea of the (*f*,*ε*)-similarity between nodes of G_A and G_B. Binary matrices $V^b(\varepsilon)$ for two values of ε are presented. Filled cells describe node-to-node assignment of G_A to G_B, which create an optimal assignment

5.2.3.4. Formulation of the multicriteria weighted graphs similarity problem (*MWGSP*)

Let us accept $SG = \{G_1, G_2, ..., G_M\}$ as a set of weighted graphs defining certain objects. Moreover, we have a weighted graph *P* that defines a certain pattern object. The problem is to find such a graph G^o from *SG* that is the most similar to *P*. We define this problem as a multicriteria weighted graphs similarity problem (*MWGSP*), which is a multicriteria optimization problem in the space *SG* with relation R_D:

$$MWGSP = (SG, F, R_D) \tag{5.24}$$

where:

$$F : SG \to R^3,\ F(G) = \left(d_S(P,G), d_{QN}(P,G), d_{QA}(P,G)\right) \tag{5.25}$$

$$R_D = \begin{Bmatrix} (Y,Z) \in SG \times SG :\ d_S(P,Y) \geq d_S(P,Z) \wedge \\ d_{QN}(P,Y) \leq d_{QN}(P,Z) \wedge \\ d_{QA}(P,Y) \leq d_{QA}(P,Z) \end{Bmatrix} \tag{5.26}$$

Domination relation R_D (Pareto relation between elements of *SG*) gives possibilities to compare graphs from *SG*. Weighted graph *Z* is more similar to *P* than *Y* if structural similarity between *P* and *Y* is not smaller than between *P* and *Z* and, simultaneously, both quantitative similarities between *P* and *Y* are not greater than between *P* and *Z*. There are many methods for solving the problem (5.24) (Eschenauer *et al.*, 1990): weighted sum (scalarization of set of objectives), hierarchical optimization (the idea is to formulate a sequence of scalar optimization problems with respect to the individual objective functions subject to bounds on previously computed optimal values), trade-off method (one objective is selected by the user and the other ones are considered as constraints with respect to the individual minima), method of distance functions in L_p-norm ($p \geq 1$) and others. We propose to use the scalar function $H(G) : SG \to R$ as a weighted sum of objectives:

$$\begin{gathered} H(G) = \alpha_1 \cdot d_S(P,G) + \alpha_2 \cdot \left(-d_{QN}(P,G)\right) + \alpha_3 \cdot \left(-d_{QA}(P,G)\right) \\ \alpha_1, \alpha_2, \alpha_3 \geq 0, \quad \alpha_1 + \alpha_2 + \alpha_3 = 1 \end{gathered} \tag{5.27}$$

Taking into account (5.27) the problem of finding the most matched G^o to pattern *P* can be formulated as follows: to determine such a $G^o \in SG$, that $H(G^o) = \max\limits_{G \in SG} H(G)$. In the last column of Table 5.1 the scalar function *H*(*G*) is defined as follows:

$$H(G) = \alpha_1 \cdot d_S(P,G) + \alpha_2 \cdot (-d_{QN}(P,G)) + \alpha_3 \cdot (-d_{QA}(P,G)) \tag{5.28}$$

where $\alpha_1 = \alpha_2 = 0.5$, $\alpha_3 = 0$, $P = G_A$, $SG = \{G_B, G_C, G_D, G_E\}$. Let us note that the best matched graph to G_A being the solution of *MWGSP* with the scalar function *H*(*G*) is G_C ($H(G^o{=}G_C)$=0.706).

Let us estimate computational complexity of the weighted graphs similarity. Let n=max{n_A,n_B}, m=max{m_A,m_B}. To compute structural similarity measure (5.17) we must first calculate matrix (5.15) and next solve the problem (5.17)-(5.20). Computation of matrix (5.15) takes a time $O(n^{2.376})$ because in practice k<<n (using matrix multiplications algorithm given by Coppersmith and Winograd with an asymptotic complexity of $O(n^{2.376})$, (Cormen, 1994)). Solving the problem (5.17)-(5.20) takes a time $O(n^3)$ using implementation of Hungarian algorithm given by Edmonds and Karp, so we obtain total complexity of these two steps $O(n^3)$. To compute nodes quantitative similarity measure d_{QN} we must first compute matrix $\overline{V} = \left[\overline{v}_{ij}\right]_{n\times n}$ in time $O(LF \cdot n^2)$ and then solve the modified problem (5.17)-(5.20) in time $O(n^3)$, so we obtain total complexity of these two steps $O(LF \cdot n^2 + n^3)$. For calculate arcs quantitative similarity measure d_{QA} we obtain complexity by analogy like for d_{QN} and we have $O(LH \cdot m^2 + m^3)$. Finally, computational complexity of total graph measure (5.27) is equal $O(LF \cdot n^2 + n^3 + LH \cdot m^2 + m^3)$.

5.2.3.5. Application of weighted graphs similarity to pattern recognition of decision situations

In the presented proposition the weighted graphs similarity approach to the identification of the decision situation is used. It consists of three stages:

1. Building weighted graphs *WGT*(*CS*), *WGD*(*CS*) and *WGT*(*PS*), *WGD*(*PS*) representing decision situations: current (*CS*) and pattern (*PS*) for topographical conditions (*WGT*) and units (potential) deploying (*WGD*);
2. Calculation of similarity measures between pairs: *WGT*(*CS*), *WGT*(*PS*) and *WGD*(*CS*), *WGD*(*PS*) for each $PS \in PDSS_{CS}$;
3. Selecting the most similar *PS* to *CS* using calculated similarity measures.

Stage 1

The first stage is to build weighted graphs *WGT* and *WGD* as follows:

$$WGT = \left\langle GT = \langle N_{GT}, A_{GT}\rangle, \{f_k^T(n)\}_{\substack{k\in\{1,\dots,5\}\\ n\in N_{GT}}} \right\rangle, \quad WGD = \left\langle GD = \langle N_{GD}, A_{GD}\rangle, \{f_k^D(n)\}_{\substack{k\in\{1,\dots,4\}\\ n\in N_{GD}}} \right\rangle$$

where *G* (*GT* or *GD*) – Berge's graphs, $G = \langle N_G, A_G \rangle$, N_G, A_G – sets of graph nodes and arcs, $A_G \subset \{\langle n, n' \rangle : n, n' \in N_G\}$. Weighted graphs *WGT* and *WGD* describe decision situations (current *CS* and pattern *PS*). Each node *n* of *GT* and *GD* describes terrain cells $(i,j)=n$ with non-zero values of characteristics defined as components of SD_{ij}^{5} from (5.1) and $\forall_{k\in\{1,\ldots,4\}} f_k^T(n)=SD_{ij}^{5,k}$, $f_5^T(n)=SD_{ij}^{5,8}$, $\forall_{k\in\{1,\ldots,3\}} f_k^D(n)=SD_{ij}^{5,4+k}$, $f_4^D(n)=SD_{ij}^{5,8}$. Two nodes $x,y\in N_{GD}$ (for $x,y\in N_{GT}$ by analogy) are linked by an arc, when cells represented by *x* and *y* are adjacent (more precisely: they are adjacent cells that take into account the direction of action, see Fig. 5.6). For example, the terrain can be divided into 15 cells (3 rows and 5 columns, left-hand side, see Fig. 5.6). The units are located in cells (denoted by circles and Xs). Structural representation of deployment of units is defined by the graph *GD*. Let us note that similar representation can be used for topographical conditions (single graph for one of the topographical information layer: waters, forests, passability or single graph *GT* for all of this information, see Fig. 5.6, right-hand side).

Stage 2

Having weighted graphs *WGD*(*CS*) and *WGD*(*PS*) (*WGT*(*CS*) and *WGT*(*PS*)) representing the current *CS* and the pattern *PS* decision situations (for units deploying) we use the procedure described in chapter 5.2.3.2 to calculate the structural and quantitative similarity measures for both graphs.

We obtain for *WGD*:

$d_S(WGD(CS), WGD(PS))=d_S^D(CS,PS)$, $d_{QN}(WGD(CS), WGD(PS))=d_{QN}^D(CS,PS)$

and for *WGT*:

$d_S(WGT(CS),WGT(PS))=d_S^T(CS,PS)$, $d_{QN}(WGT(CS),WGT(PS))=d_{QN}^T(CS,PS)$.

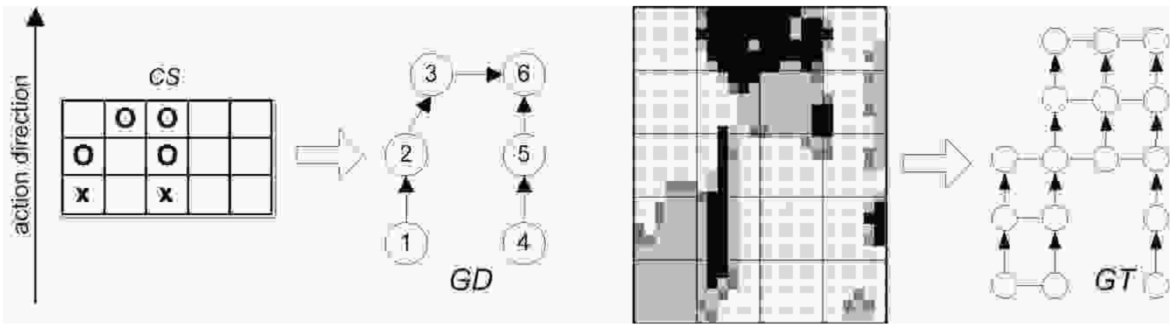


Fig. 5.6. Deployment of units and their structural (graph *GD*) representation (left-hand side) and terrain covering (growth) and its structural (*GT*) representation (right-hand side). Circles (O) and crosses (X) describe two types of units

Stage 3

We formulate problem (5.24), separately for *WGT* and *WGD*, where: *SG*:=*PDSS*, *F*(*G*):=*F*$_D$(*PS*), $d_S(P,G):=d_S^D(CS,PS)$, $d_{QN}(P,G):=d_{QN}^D(CS,PS)$ for *WGD* and $F(G)$:=$F_T(PS)$, $d_S(P,G):=d_S^T(CS,PS)$, $d_{QN}(P,G):=d_{QN}^T(CS,PS)$ for *WGT*. Next, we define the scalar functions (5.27) to solve the problem (5.24) for *WGD* and *WGT*:

$$H_D(\cdot)=\alpha_1\cdot d_S^D(\cdot,\cdot)+\alpha_2\cdot(-d_{QN}^D(\cdot,\cdot)) \tag{5.29}$$

and

$$H_T(\cdot)=\gamma_1\cdot d_S^T(\cdot,\cdot)+\gamma_2\cdot(-d_{QN}^T(\cdot,\cdot)) \tag{5.30}$$

Having $H_D(PS)$ and $H_T(PS)$ we can combine these criteria (as in (5.27)) or set threshold values and select the most matched pattern situation to the current one. This process requires "rich" knowledge base of pattern situations in order to better learn of the *MWGSP* algorithm.

Concept of *MWGSP* can be also used to estimate real realization of course of action (Antkiewicz *et al.*, 2009d):

- we define formation of conflict side using network representation from (5.13);
- taking into account (5.13) we define pattern formation *PF* of conflict side; the *PF* is predicted (or demanded) formation which should be achieved after actions;
- after simulation of course of actions s_i (i=1,..., N) we obtain $ASF(s_i)$ formation for each i=1,..., N;
- we calculate structural $d_S(PF,ASF(s_i))$ and quantitative $d_{QN}(PF,ASF(s_i))$ similarity measures between *PF* and $ASF(s_i)$ using procedure described in chapter 5.2.3.2;
- we can solve *MWGSP* defined in chapter 5.2.3.4 to find the best s_i^* course of action from the point of view of its formation similarity to demanded *PF* formation.

5.2.3.6. Numerical example

An example of using the approach presented in chapter 5.2.3.5 to find the most matched pattern decision situation to the current one is presented in Fig. 5.7 and in Table 5.2. Results of calculations $H_D(PS)$ are presented for each $PS\in PDSS_{CS}=\{PS_1,...,PS_8\}$. Only function $f_4^{D(CS)}(n)=SD_{ij}^{5,8}$ ($f_4^{D(PS)}(n)$ for pattern *PS*) is used from *WGD* to compute quantitative similarity of nodes (see chapter 5.2.3.2) because all units have the same type. Thus, vector $v(WGD(CS),WGD(PS))$ of matrices has one component $V_1=[v_{ij}(1)]_{|N_{GD(PS)}|\times|N_{GD(CS)}|}$. Function $f_4^{D(CS)}(n)$ describes coordinates of node n (the left-lower cell has coordinates (1,1)). The norm from (5.21) has the form of: $\left\|f_4^D(i)-f_4^D(j)\right\|_{p=2}=\left(\sum_{r=1}^{2}\left|f_{4,r}^D(i)-f_{4,r}^D(j)\right|^2\right)^{1/2}$ and it describes the geometric distance between nodes $i\in N_{GD(PS)}$ and $j\in N_{GD(CS)}$. Let us note that for weights $\alpha_1=0$, $\alpha_2=1$ values in Table 5.2 (for the row PS_i) describes $d_{QN}^D(CS,PS_i)$

and for $\alpha_1 = 1$, $\alpha_2 = 0$ describes $d_S^D(CS, PS_i)$. The best matched *PS* to *CS* is *PS*$_2$ (taking into account d_S^D and d_{QN}^D).

The process of optimal selection of weights can be organized as follows: we build a learning set $\{CS_i, PDSS_i\}_{i=1,\ldots,LS}$ and for different values of weights experts estimate whether, in their subjective opinion, CS_i is similar to $PS^* \in PDSS_i$ determined from the procedure. The combination of weight values, which are indicated by majority of experts, is the optimal combination.

Some other applications of the *MWGSP* problem are presented in chapter 6.3.

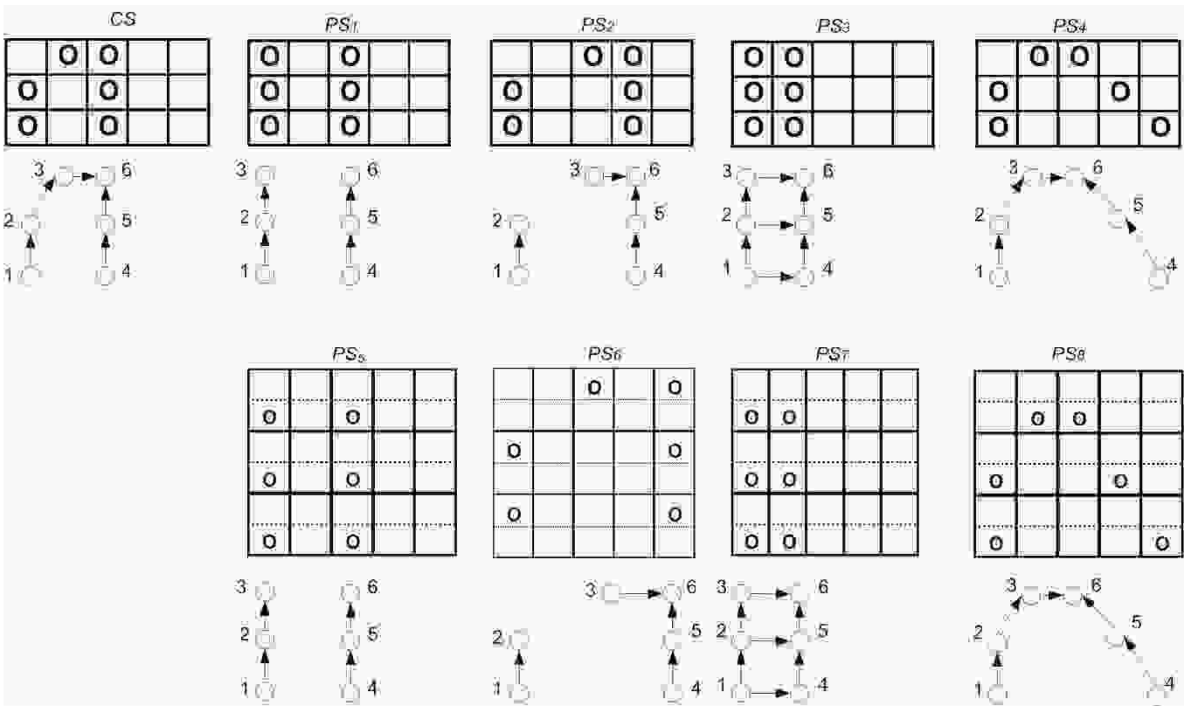


Fig. 5.7. The current situation *CS* with graph *GD*(*CS*) and eight pattern situations *PS*$_i$ (*i*=1,…,8) with graphs *GD*(*PS*$_i$) describing structure of units deployment. Patterns 1-5, 2-6, 3-7 and 4-8 have the same structure, but cells for patterns 5,..,8 have a greater size than for patterns 1,…,4

Table 5.2. Values of the scalar function $H_D(PS_i)$ combining structural (weight α_1) and quantitative (weight α_2) similarity measures between *GD*(*CS*) and *GD*(*PS*$_i$) from Fig. 5.7. The best (maximal) values in the columns are denoted in bold

Pattern	*Weights* (α_1; α_2)				
PS$_i$	(0; 1)	(0.33; 0.67)	(0.5; 0.5)	(0.67; 0.33)	(1; 0)
PS$_1$	**-0.094**	**0.283**	0.463	0.800	**1.527**
PS$_2$	-0.370	**0.283**	**0.593**	**0.870**	1.504
PS$_3$	-0.478	0.157	0.360	0.726	1.254
PS$_4$	-0.233	0.176	0.467	0.827	**1.527**
PS$_5$	-0.474	0.120	0.461	0.824	**1.527**
PS$_6$	-0.706	0.032	0.378	0.761	1.504
PS$_7$	-0.63	0.070	0.279	0.631	1.254
PS$_8$	-0.508	0.047	0.415	0.793	**1.527**

5.3. Decision Automata for a March

In chapter 5.2 elements of decision automata for an attack, which replaces the commander at the battalion level, have been described. In this chapter we present the decision automata for marching which execute two main processes (Tarapata, 2007e): the march planning process and direct march control. The march planning process relating to the automata includes the determination of: march organization, paths for units and detailed march schedule for each unit in the column. The direct march control process contains such phases like command, reporting and reaction to fault situations during the march simulation. The automata is implemented in the ADA language and it represents a commander of battalion level (the lowest level of trainees is brigade level). It is a component of distributed interactive simulation system *SBOTSS Zlocien* for *CAXes* (*Computer Assisted Exercises*) (Najgebauer, 2004a; 2004b). Some of the applications are presented in chapter 6.1.

5.3.1. The March Planning Process

5.3.1.1. Description of the problem

The march planning process relating to the automata contains the determination of such elements as: march organization (units order in the march column, count and stopping points), paths for units and detailed march schedule for each unit in the column. Algorithms, which carry out the decision planning process described below, are presented in chapter 5.3.3.

The decision process for the march starts at the moment *t*, when the battalion *id* receives the march order *SO*(*id*, *t*) from a superior (brigade) unit. The structure of the *SO*(*id*, *t*) is as follows:

$$SO(id,t)=\left(t_0(id,t),\, t_S(id,t),\, MD(id,t)\right) \tag{5.31}$$

where: *SO*(*id*, *t*) – superior order to march for battalion *id*; $t_0(id,t)$ – readiness time for the unit *id*; $t_S(id,t)$ – starting time of the march for the unit *id*; $MD(id,t)$ – detailed description of the march order. Definition of the $MD(id)$ (we omit *t*) is as follows:

$$MD(id)=\left\langle S(id),D(id),\, RP(id),IP(id)=\left(in_p(id),it_p(id)\right)_{p=\overline{1,NIP}}\right\rangle \tag{5.32}$$

where: $S(id),D(id)$ – source and destination areas for *id*, respectively; $RP(id)$ – the rest area for the *id* unit (after twenty-four-hours of marching), optional; $IP(id)$ – vector of checkpoints for the *id* unit (march route must cross these points), $in_p(id)$ – the *p*-th checkpoint, $in_p(id)\in W_1\cup W_2$, W_1, W_2 defined in chapter 2.3, $in_1(id)=PS(id)$

is the starting point of the march (at this point the head of the marching column is formed) and it is required, other checkpoints are optional, $it_p(id)$ – time of achieving the p-th checkpoint (optional); NIP – number of checkpoints. After the id unit (battalion) receives the brigade commander's order to march, the decision automata starts planning the realization of this task. Taking into account $SO(id,t)$, for each unit id' (of company level and equivalent) directly subordinate to id the march order, $MDS(id')$ is determined as follows:

$$MDS(id') = \langle S(id'), D(id'), PS(id'), PD(id'), RP(id'), \mu(id', S(id'), D(id')) \rangle \tag{5.33}$$

where: $S(id'), D(id')$ – source and destination areas for id', respectively, $S(id') \subset S(id)$, $D(id') \subset D(id)$; $RP(id')$ – rest area for the id' unit (after twenty-four-hours of marching), $RP(id') \subset RP(id)$, optional parameter; $PS(id')$ – starting point for the id' unit, the same for all $id' \in id$ and $PS(id') = in_1(id) \in W_1 \cup W_2$; $PD(id')$ – ending point of the march for the id' unit, the same for all $id' \in id$ and $PD(id') \in W_1 \cup W_2$; $\mu(id', S, D)$ – the route for the unit id' from the region $S(id')=S$ to region $D(id')=D$, $\mu(id',S,D) = \left(w(id',m), v(id',m)\right)_{m=\overline{1,LW(\mu(id',S,D))}}$, $w(id',m)$ – the m-th node on the path for id', $w(id',m) \in W_1 \cup W_2$, $S, D \subset W_1 \cup W_2$ and $w(id',1) \in S$, $w\left(id', LW\left(\mu(id',S,D)\right)\right) \in D$; $LW(\mu(id',S,D))$ – number of nodes (squares or crossroads) on the path $\mu(id',S,D)$ for id' unit; $v(id',m)$ – velocity of the id' unit on the arc starting in the m-th node.

5.3.1.2. Models of movement plans

The movement models define following movement plans:

(a) from point (region) to point (region);
(b) visiting selected points (regions);
(c) omitting selected points (regions, obstacles);
(d) inside or outside selected region;
(e) off-roads only;
(f) on-roads only;
(g) combined on- and off-roads.

They use following criterions for paths planning: time minimization, distance minimization, camouflage degree maximization. We define general problem for finding the best route which includes problems (a)-(g). We formulate problem for extreme path finding for id unit which realize movement plans (a)-(g) as follows:

in the network $S^z = \langle G^z, \Psi_1(t) \cup \Psi_2(t), \zeta_2(t) \cup \{l_1, l_2, l_3\} \rangle$ (defined in chapter 2.3) to find a such path $\mu^*(id,S,D) \in M(id,S,D)$, for which

$$K\left(\mu^*(id,S,D)\right) = \underset{\mu(id,S,D) \in M(id,S,D)}{extr} K\left(\mu(id,S,D)\right) \tag{5.34}$$

where: $K(\mu(id,S,D))$ – "cost" of the path $\mu(id,S,D)$,

$$K(\mu(id,S,D)) = \sum_{m=1}^{LW(\mu(id,S,D))-1} l\left((w(id,m),w(id,m+1))\right) \tag{5.35}$$

$M(id,S,D)$ – set of acceptable paths from the region S to the region D for *id* unit,
$l\left((w(id,m),w(id,m+1))\right)$ – arc $(w(id,m),w(id,m+1))$ cost function.

It is important to note that path $\mu(id,S,D)$ may consist of sequences of nodes from $Z_1(t)$ and $Z_2(t)$ defined in chapter 2.3 (when we accept descending from the road on the squares (if it is possible) and vice versa), see Fig. 5.8.

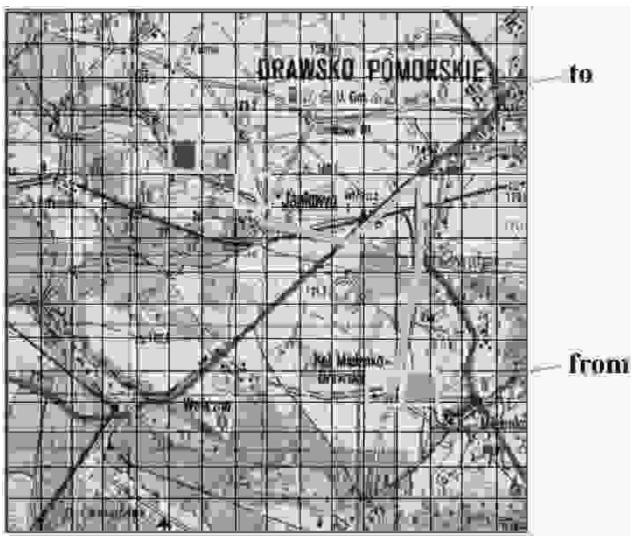


Fig. 5.8. The idea of hybrid path in the *Zlocien* system. The path consist of 5 (2+3) squares and 5 parts of road

If we want:

- to minimize movement time, then in (5.35) $l\left((\square,\square)\right)=l_1\left((\square,\square)\right)$, and in (5.34) *extr=min,* where l_1 function defined by (2.15);
- to minimize geometrical length (distance) of the path then in (5.35) $l\left((\square,\square)\right)=l_2\left((\square,\square)\right)$ and in (5.34) *extr=min*, where l_2 function defined by (2.20);
- to maximize degree of camouflage for determined path then in (5.35) $l\left((\square,\square)\right)=l_3\left((\square,\square)\right)$ and in (5.34) *extr=max*, where l_3 function defined by (2.21).

Moreover, depending on kind of the movement plan (a)-(g), we define the set $M(id,S,D)$ of acceptable paths in the different way:

- for the case (a):

$$M(id,S,D)=\left\{\mu(id,S,D)=(w(id,m),v(id,m)) : S\subset W^z, D\subset W^z\right\} \tag{5.36}$$

but if we determine path from the point (node) to the point (node) then $\overline{\overline{S}} = 1$ and $\overline{\overline{D}} = 1$. It is important to emphasize, that for each $m \in \{1,...,LW(\mu)-1\}$, $v(id,m) = v^{slowd}\left(id,\left(w(id,m),w(id,m+1)\right)\right)$, where $v^{slowd}(\bullet,\bullet)$ described by (2.17).

- for the case (b):

$$M(id,S,D) = \left\{\mu(id,S,D) = \left(w(id,m),\Box\right) : \underset{a\in P(id)}{\forall}\ \underset{m\in\{1,...,LW(\mu)\}}{\exists} w(id,m) = a\right\} \tag{5.37}$$

where $P(id) \subset W^z$ describes subset of W^z which must belong to the path μ;

- for the case (c):

$$M(id,S,D) = \left\{\mu(id,S,D) = \left(w(id,m),\Box\right) : \underset{a\in NP(id)}{\forall} \sim \underset{m\in\{1,...,LW(\mu)\}}{\exists} w(id,m) = a\right\} \tag{5.38}$$

where $NP(id) \subset W^z$ describes subset of W^z which path μ must omit;

- for the case (d1):

$$M(id,S,D) = \left\{\mu(id,S,D) = \left(w(id,m),\Box\right) : \underset{m\in\{1,...,LW(\mu)\}}{\forall} w(id,m) \in OW(id)\right\} \tag{5.39}$$

where $OW(id) \subset W^z$ describes connected subset of W^z inside which the path μ must cross;

- for the case (d2):

$$M(id,S,D) = \left\{\mu(id,S,D) = \left(w(id,m),\Box\right) : \underset{m\in\{1,...,LW(\mu)\}}{\forall} w(id,m) \notin OZ(id)\right\} \tag{5.40}$$

where $OZ(id) \subset W^z$ describes subset of W^z outside which the path μ must cross;

- for the case (e):

$$M(id,S,D) = \left\{\mu(id,S,D) = \left(w(id,m),\Box\right) : \underset{m\in\{1,...,LW(\mu)\}}{\forall} w(id,m) \in W_2\right\} \tag{5.41}$$

- for the case (f):

$$M(id,S,D) = \left\{\mu(id,S,D) = \left(w(id,m),\Box\right) : \underset{m\in\{1,...,LW(\mu)\}}{\forall} w(id,m) \in W_1\right\} \tag{5.42}$$

- for the case (g): the same like for the case (a).

It is possible to define set $M(id,S,D)$ which is the common part of sets above defined. For example, we may have the following requirements for the path: it must cross from the point (square) to the selected region, it must omit selected points, it must cross inside selected region and it must be "off-roads". This situation may concern e.g. movement path for attacking company, which must move from the occupied region to the region occupied by the opposite unit, inside the zone of attack, omitting in this zone, for example, recognized minefields. Definition of the set $M(id,S,D)$ in this situation is following (we use (5.36), (5.38), (5.39) and (5.42)):

$$M(id,S,D)=\{\mu(id,S,D)=(w(id,m),\square):\underset{m\in\{1,\ldots,LW(\mu)\}}{\forall} w(id,m)\in OW(id)\wedge$$
$$\wedge\, w(id,m)\notin NP(id)\wedge w(id,m)\in W_1\wedge\left(\overline{\overline{S}}=1\wedge\overline{\overline{D}}\geq 1\right)\} \qquad (5.43)$$

5.3.1.3. March organization determination

March organization includes the determination of such elements as: number of columns, order of units in march columns and number and place of stops.

Number (#) of columns results from tactical rules and depends on the tactical level of the unit: for the battalion level #columns=1, for the brigade level #columns∈{1,2,3}; for the division level #columns∈{3,4,5}. In Fig. 5.9 each brigade has a single march column consisting of two battalions equipped with 4 companies each one; unit 111 is the head of the 1st brigade column (and simultaneously it is the head of the 1st battalion column); *dc* – distance in battalion column between companies, *lc* – company column length; *db* – distance in brigade column between battalions. Order of units in march column results from tactical rules as well (algorithm *Units_Order_In_March_Column_Determ*(*id'*), see Table 5.3).

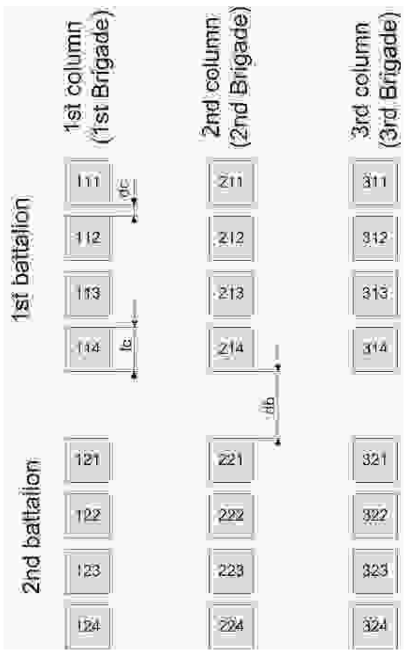


Fig. 5.9. Example of march organization in three columns

Number of stops $c_{stops}(id)$ is calculated as follows (algorithm *Number_of_Stops_Determ*(*id'*), see Table 5.3):

$$c_{stops}(id)=\max\left\{\left\lfloor\frac{\left(t_D(id,t)-t_S(id,t)-t_{rest}(id)\right)\cdot v_{avg}(id)-L_{path}(id)}{v_{avg}(id)\cdot\left(t_{stop}(id)+\Delta s\right)}\right\rfloor,0\right\} \qquad (5.44)$$

where: $t_D(id,t)$ – demanded ending time of the march for the *id* unit, $t_S(id,t)$ – starting time of the march for the *id* unit (as in (5.31)), $t_D(id,t) > t_S(id,t) \geq 0$, $t_{rest}(id)$ – duration time of the rest for the *id* unit, $v_{avg}(id)$ – average march velocity for the *id* unit, $L_{path}(id)$ – length of the path determined for the *id* unit (in km), $t_{stop}(id)$ – duration time of the stop for the *id* unit, Δs – time interval between stops. In practice, values of parameters are as follows: $t_{rest}(id) \approx$24h, $v_{avg}(id) \in [30 \div 40]$ km/h, $t_{stop}(id) \approx 1$ h, $\Delta s \in [3,4]$ h.

Place of stops are fixed after path determination and algorithm *Place_Of_Stops_Determ(id′)* (see Table 5.3) takes into account $c_{stops}(id)$ and the *FCam* function (see Table 2.1) to find optimal positions of stops.

5.3.1.4. Detailed march schedule determination

Detailed movement schedule for *id′* unit is defined as follows (procedure *Detailed_Schedule_Determ*(*id*′) in Table 5.3):

$$H(id',t_0) = \langle S, D, \mu(id',S,D), T(id',S,D) \rangle \tag{5.45}$$

where: *t*0 – starting moment of the schedule realization; $T(id',S,D)$ – vector of moments of achieving nodes on the path, $T(id',S,D) = \langle t(id',m) \rangle_{m=\overline{1,LW(\mu(id',S,D))}}$, $t(id',m)$ – moment of achieving the *m*-th node on the path,

$$t(id',m) = t_0 + \sum_{j=1}^{m-1} \frac{L(w(id',j), w(id',j+1))}{v(id',j)} \tag{5.46}$$

and *L*(*w*(*id′*,*j*),*w*(*id′*,*j*+1)) describes the geometric distance between the *j*-th and the (*j*+1)-st nodes on the path, *LW(μ(id′,S,D)* – number of nodes on the path for *id′* unit. After determining *MDS*(*id′*) each unit *id′* is subordinate to battalion *id*, the order is sent by automata to each of the *id′* units. The idea of determining the march route for unit *id* is presented in Fig. 5.10. In this figure we have three checkpoints: P_1=*PS*, P_2 and P_3=*PD* (the path for all units must follow these points). P_1 is the starting point of the march (in this point the head of the march column consisting of three units is formed), P_3 is the end point of the march (at this point the march column is resolved), P_2 is the intermediate point of the march. The path between P_1 and P_3 is common for all units, however each unit has its own path from subarea of *S* to P_1 and from P_3 to subarea of *D*.

In general, the automata uses two types of categories of criteria for synchronous movement scheduling of the *K* object (unit) columns defined in chapter 4.2.1.1: (4.14) and (4.15). Taking into account that unit *id* is equivalent to the *k*-th column we can write the following equivalence between notations being used in chapters 4.2-4.3 and this chapter:

$$v^k_{i^r(k),i^{r+1}(k)} \equiv v(k,r),\ i^r(k) \equiv w(k,r),\ d_{i^r(k),i^{r+1}(k)} \equiv L\big(w(k,r),w(k,r+1)\big),\ r_p(k) = in_p(id).$$

One of the formulations of the optimization problem for movement synchronization scheduling of *K* objects used in the automata is presented in chapter 4.3.1 and methods for solving it in chapter 4.3.2. Theses methods are implemented in the automata with a common name *March_Schedule_Determ*(*id*') (see Table 5.3). One of the methods being used inside the previous one is *Paths_Determ*(*id*').

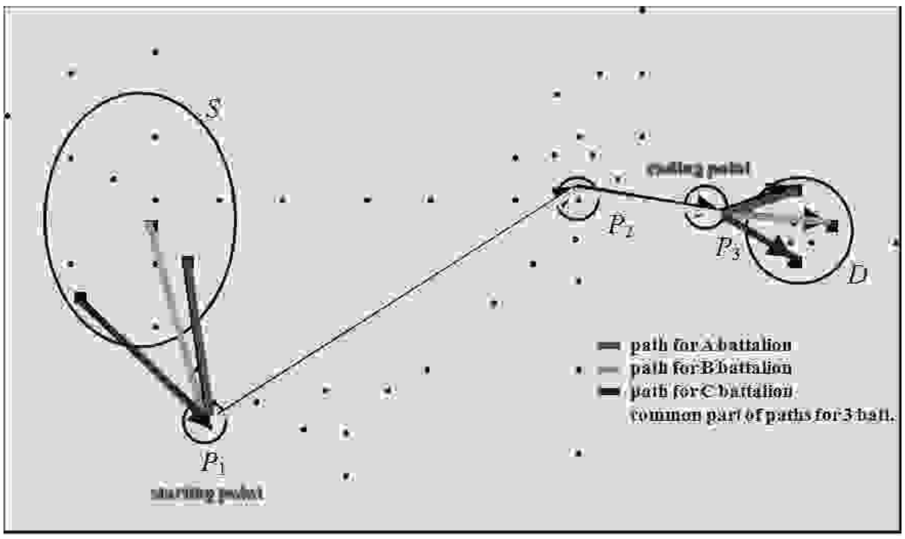


Fig. 5.10. An example of a march route (path) for three units $id' \in id$ (filled squares) from the *S* source area to the *D* destination area (dots represent crossroads from a digital map)

5.3.2. The Direct March Control

The direct march control process contains such phases as: command, reporting and reaction to fault situations during march simulation (Tarapata, 2007e). Let us remember that the automata replaces the battalion commander and manages subordinate units (company or/and platoons and equivalent).

In the movement simulation we "see" the units column on the road twofold: (a) as occupying arcs (part of the roads) and nodes (crossroads) of the Z_2 network (from equation (2.4)), (b) as a sequence of squares of the Z_1 network (from (2.4)) by which the arc crosses. In case (a) we move the head and the tail of the column and we register arcs of Z_2 in which the head and the tail are located with the degrees of crossing these arcs. In case (b) we locate the head and the tail of the column on the squares of the Z_1 network and we move the "sequence" of these squares (from the head to the tail).

Movement of the unit on the road (deployed in the column) is done by determining the sequence of nodes (crossroads) and arcs (part of the roads) of the Z_2 network and next we execute the movement from crossroad/square to crossroad/square (procedure *Simulate_Unit_Movement*(*id'*) in Table 5.3, see also chapter 6.1.3).

5.3.2.1. Identifying fault situations during a march simulation and automata reactions

The automata for marching on the battalion level reacts to fault situations during the march simulation presented below (procedure *React_To_Fault_Situations*(*id'*), see Table 5.3):

1. Current velocity of a subordinate unit differs from the scheduled velocity;

Reaction: (a) If a unit is the head of the column and it does not move at planned velocity then increase the velocity (in case of delay) or decrease it (in case of acceleration); (b) If a unit is not the head of the column then adapt the velocity to the velocity of the preceding unit.

2. Reaching critical fuel level in one of the subordinate units;

Reaction: Report to the automatic commander. Attempt refuelling at the next stop or refuel as soon as possible.

3. Detection of an opponent unit;

Reaction*:* If the opponent forces are overwhelming (opponent combat potential is greater than the threshold value) and distance between own and opponent units is relatively small then the unit is stopped, it goes to defensive position and reports to the commander. Otherwise, reports only to the commander.

4. Detection of a minefield;

Reaction: Stop and report to the commander.

5. Loss of capability to execute march (destruction of part of the march route (e.g. bridge, river crossing), other cause of impassability);

Reaction: (a) If the route is impassable due to destruction of a part of the march route then attempt to find a detour. Report to commander; (b) If other cause of impassability then take defensive position and report to the commander.

6. Contamination of part of the march route or a subordinate unit;

Reaction: Report to commander. If degree of contamination is low then run chemical defence and continue a march, otherwise try to exit from contaminated area.

Situations which require reporting to the superior of the battalion (procedure *Report_To_Commander*(*id'*) in Table 5.3):

(a) achieving checkpoints, stop area or rest area;
(b) slowing down velocity which causes delays;
(c) encountering contamination;
(d) encountering a minefield;
(e) reaching 75% and 50% of standard fuel level;
(f) capability loss of march execution (reporting the cause of capability loss);
(g) detection of opponent units.

5.3.2.2. Velocity calculation

The important problem during the simulation is to set the current velocity of the unit *id* because of the necessity for synchronous movement of many columns. The procedure of the velocity setting (procedure *Adapt_March_Velocity*(*id'*), see Table 5.3) inside the *n*-th square consists of two cases: (a) when the unit *id* is not engaged in combat in the *n*-th square; (b) when the unit *id* is engaged in combat in the *n*-th square.

In case (a) the current velocity $v_{cur}(id,n)$ of the unit *id* in the *n*-th terrain square is calculated as follows:

$$v_{cur}(id,n)=\min\{v^{slowd}(id,n),\, v_{dec}(id,n)\} \tag{5.47}$$

where: $v^{slowd}(id,n)$ – maximal velocity of the unit *id* in the *n*-th square taking into account topographical conditions, $v^{slowd}(id,n)$ is equivalent to $v^{slowd}(id,(n,n))$ from (2.17), $v_{dec}(id,n)$ – velocity resulting from the commander decision and equals $v(id',j)$ in (5.46), $id' \equiv id$, $j \equiv n$.

If the unit *id* is the head of column and it does not move with planned velocity $v_{dec}(id,n)$ then the velocity is increased (in case of delay) or decreased (in case of acceleration). If the unit *id* is not the head of the column then the velocity of the unit *id* is adapted to the velocity of the preceding unit. This movement method is known as follow-the-leader (e.g. in Fig. 5.9 the leader of the 1st brigade is unit 111).

In case (b) the current velocity $v_{cur}(id,n)$ of the unit *id* in the *n*-th square is calculated as follows:

$$v_{cur}(id,n)=\min\left\{f\left(v^{slowd}(id,\square),U_A,U_B,dist\right),v_{dec}(id,n)\right\} \tag{5.48}$$

where: $f(\square,\square,\square,\square)$ – function describing the velocity in the square dependent on $v^{slowd}(id,\square)$, potentials of the unit *id* of side A (U_A) and B (U_B) which are fighting and distance (*dist*) between fighting sides.

5.3.2.3. Fuel consumption calculation

Fuel consumption $FC(id,veh,u)$ (procedure *Fuel_Consumption_Determ*(*id'*) in Table 5.3) on the *u* part of the path for the type of vehicle *veh* belonging to the *id* unit is calculated as follows:

$$FC(id,\ veh,\ u)=FLen(u)\cdot FCC(u,veh)\cdot\frac{NFC(veh)}{100}\cdot N(id,veh) \tag{5.49}$$

where: $FLen(u)$ describes the length of the *u* part of a path (see Table 2.2), $FCC(u,veh)$ – fuel consumption coefficient for the *u* part of the path and for the vehicle type *veh*, $NFC(veh)$ – normative average fuel consumption for the *veh* type of vehicle (per 100km), $N(id,veh)$ – number of vehicles of *veh* type in the *id* unit. Fuel consumption coefficient *FCC* is calculated as follows:

$$FCC(u,veh) = (1.0 + MTC(veh)) \cdot (1.0 + UC(u)) \tag{5.50}$$

where $MTC(veh)$ describes the mechanical-tactical coefficient and $UC(u)$ – utilization coefficient, $veh \in K\_Veh$ resulting from logistic calculations (see details in (Tarapata, 2007e)).

5.3.3. Automata Implementation

The automata is implemented in ADA language and it represents a part of an automatic commander on the battalion level (Najgebauer *et al.*, 2007b; Tarapata, 2007e). They realize their own tasks and pass on tasks to subordinate units. Simulation objects and their methods are managed by a dedicated simulation kernel (extension of ADA language). Object methods are divided into two sets: (1) non-simulation methods – designed in order to set and get values of attributes, specific calculations and database operations; (2) simulation methods – prepared for synchronous ("wait-for" methods) and asynchronous ("tell" methods) data sending. The simulation kernel is an object package based upon a permanent process (low level ADA language task). The simulation event is stored in one of the data structures: linked list ($O(n)$ complexity) or effective *BST* tree ($\log_2(n)$ complexity). Events are sorted in chronological order resulting from timestamps (Pierzchała, 2005). In Fig. 5.11 and Fig. 5.12 general diagrams of the simulation kernel and other important associated objects are shown.

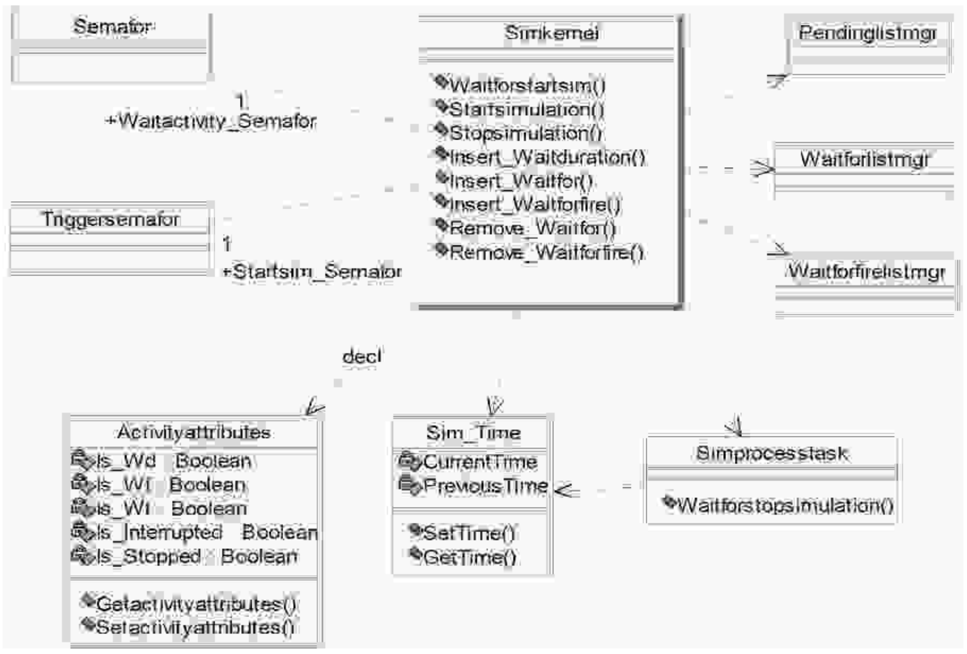


Fig. 5.11. Class diagrams of simulation kernel package (Najgebauer *et al.*, 2005)

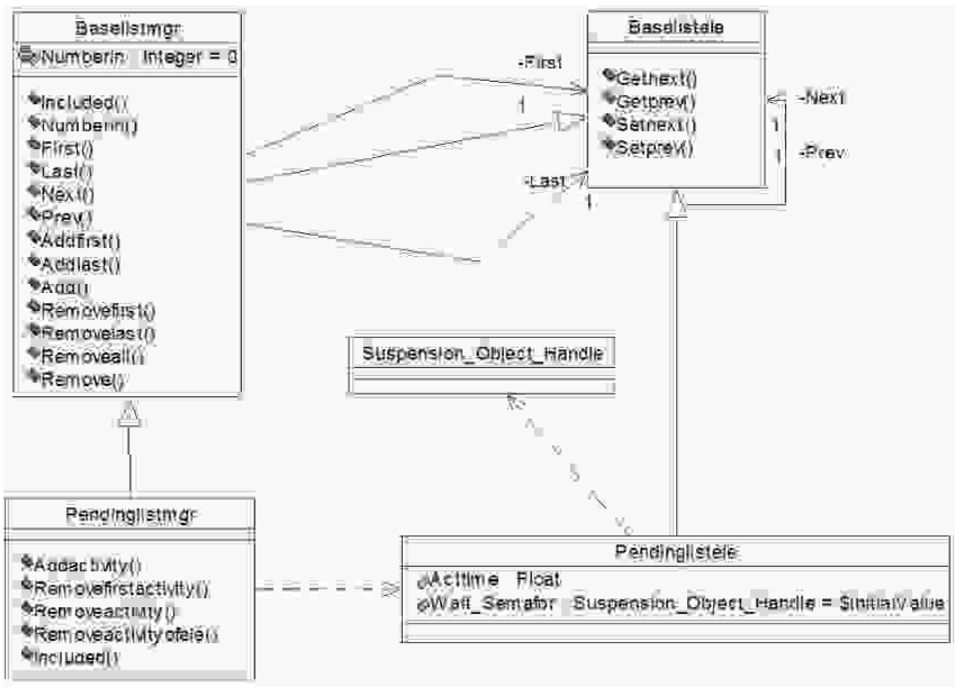


Fig. 5.12. Diagram of classes associated with the simulation kernel package (Najgebauer *et al.*, 2005)

Procedures implemented and used for decision planning and direct march control processes are presented in Table 5.3.

Some simulation methods for movement of individual and group objects as well as the method of cooperative movement simulation are presented in chapter 5.4. Moreover, a case study is presented in chapter 6.1.

Table 5.3. Procedures implemented and used for decision planning and direct march control processes in the march automata

Procedures implemented and used for each unit id′∈id for the decision planning process	*Procedures implemented and used for each unit id′∈id for the direct march control process*

Units_Order_In_March_Column_Determ(id′)
Column_Length_Determ(id′)
Number_of_Stops_Determ(id′)
Place_Of_Stops_Determ(id′)
Ending_Point_PD_Determ(id′)
March_Schedule_Determ(id′)
  Paths_Determ(id′)
    Path_ S_To_PS_Determ(id′)
    Common_Path_PS_To_PD(id′)
    Path_ PD_To_D_Determ(id′)
  Detailed_Schedule_Determ(id′)
	March_Simulation(id′)
	  Simulate_Unit_Movement(id′)
	  React_To_Fault_Situations(id′)
	  Fuel_Consumption_Determ(id′)
	  Adapt_March_Velocity(id′)
	  Report_To_Commander(id′)

5.4. Methods for Movement Simulation of Individual and Group Objects

5.4.1. Method for Movement Simulation of Individual Objects

Presented here are examples of movement simulation of military objects, which are carried out in the environment of a simulation object-oriented language MODSIM II (Modsim, 1995). Therefore, we consequently use the notation of this language. Each of the military objects may be considered as a separate MODSIM object:

-
  ```
  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 the military unit in
                (Tarapata, 2000b; 2000d))
       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 a 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;
```

The **VehicleObj** object contains attributes of the military object, as information indispensable considering terrain traffic possibility by this object, etc.

The **Wsp** record 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 the size of the node) and links.

The **DynVehicleObj** object describes a military object containing, additionally, the possibility of moving and imaging, and inheriting both from **VehicleObj** and **DynImageObj**.

The **DynImageObj** object (Modsim, 1995; Simgraphics, 1995) is the standard object of the SIMGRAPHICS II and describes the dynamic graphical object:

```
DynImageObj=OBJECT(ImageObj,MovingObj,RotatingObj,ScalingObj);
          . . .  fields and methods (see (Simgraphics, 1995, pp. 192-194))
END OBJECT;
```

This object may be drawn, moved, scaled and rotated with respect to simulation time. In this connection the **DynVehicleObj** object has the same properties because it inherited from the **DynImageObj**.

The most important properties of the **DynVehicleObj** object are presented below:

- inherited from **MovingObj**:

  FIELDS:

  * *Course* – actual course (direction) of the 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 the direction in which the object travels;

  * ASK METHOD *SetSpeed(...)* – sets the speed of the object;

  * TELL METHOD *MoveTo(...)* – moves the object to a specified point. The method stops when the object arrives at its destination.

  * TELL METHOD *FollowPath(...)* – moves the object along a path defined by an array of points. This method stops when the object arrives at the last point of the array. To stop it from continuing we should use *Interrupt*.

- inherited from **RotatingObj**:

  FIELDS:

  * *RotationSpeed* – actual speed of rotation in radians per seconds;

  * *RotatingTo* – TRUE if the object is actually rotating;

  METHODS:

  * ASK METHOD *SetRotationSpeed(...)* – sets the speed of the 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 unit of time. For example, with the scale of 1.0, the object becomes twice as big after 1 unit of time, 3 times as big after 2 unit of time, etc.;

* TELL METHOD `ScaleTo(...)` – synchronic scaling of the object to the point defined as the method parameter with speed ScaleSpeed;

- inherited from **`DynamicObj`** (which was 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) which is called automatically from the runtime library;
  * ASK METHOD `StopMotion` – stops an object from moving. `DynamicUpdate` method no longer is invoked from the runtime library;
  * ASK METHOD `DynamicUpdate(IN currTime, elapsedTime : REAL)` – called periodically by the timing routine to update animation.

Animation (moving) of the **`DynImageObj`** object type can be done in two ways. The first way is to set the object fields `Course` and `Speed` and invoke the `StartMotion` method of this object. It causes the object movement with fixed attributes. The second way 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 the destination point.

The fields and methods added by **`DynVehicleObj`** are the following:

FIELDS:

* `CurrNode;`
* `Path;`

METHODS:

* ASK METHOD `SetCurrNode(...);`
* ASK METHOD `FindPath(...);`
* TELL METHOD `MoveVehicle(...).`

The `CurrNode` field contains information about the network node lastly achieved by the object. The `Path` field contains an array of node 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 sets the path for an object. The result is an array of node numbers belonging to the path from the starting node to the ending node 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 5.1.

Example 5.1

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

BEGIN
1    ASK SELF TO DisplayAt(ASK NodeS Translation.x,
                          ASK NodeS Translation.y);
2   WHILE (i < > HIGH(Path)+1) AND (NOT exit)
3        INC(i);
4        IF  i < HIGH(Path)
5            link := ASK NetWindow TO GiveLink(
             ASK NetWindow TO GiveNode(ASK SELF  Path[i]),
             ASK NetWindow TO GiveNode(ASK SELF  Path[i+1]));
6            IF link <> NILOBJ
             { object moving }
7                NodeD := ASK link Destination;
8                NodeS := ASK link Source;
9                xs:=ASK NodeS Translation.x;
10               ys:=ASK NodeS Translation.y;
11               xd:=ASK NodeD Translation.x;
12               yd:=ASK NodeD Translation.y;
13               ASK SELF TO SetRotationSpeed(RotationSpeed);
14               WAIT FOR SELF TO RotateTo(ATAN2(ys-yd,xs-xd)+pi);
15               ON INTERRUPT
16                   IF SELF<>NILOBJ
17                       DISPOSE(SELF);
18                   END IF;
19                   exit:=TRUE;
20               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;
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 the problem described in chapter 5.3.2.2. Invoking of the method to start a synchronous object movement to the specified point (node) is presented in line 22. Independently of this, objects may be moved by means of the `StartMotion` method (see description earlier presented).

The full invoking of an object movement may resemble that in Example 5.2.

Example 5.2

```
             .
             .
             .
      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();
END METHOD;
```

5.4.2. Method for Movement Simulation of Group Objects

A method of movement simulation for grouped objects is strictly related to the movement of individual objects. An example of a grouped object is a column (convoy) of individual objects. In this case, movement of these objects may resemble that in Example 5.3.

Example 5.3

```
              .
              .
              .
   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();
END METHOD;
```

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

5.4.3. Method for Cooperating Objects Movement Simulation and Management

To plan and control *K* units movement during simulation described in chapters 5.3.1 and 5.3.2 the *Movement Synchronization Manager* (*MSM*) has been proposed (Tarapata, 2007e; 2010b) and its idea is presented in Fig. 5.13.

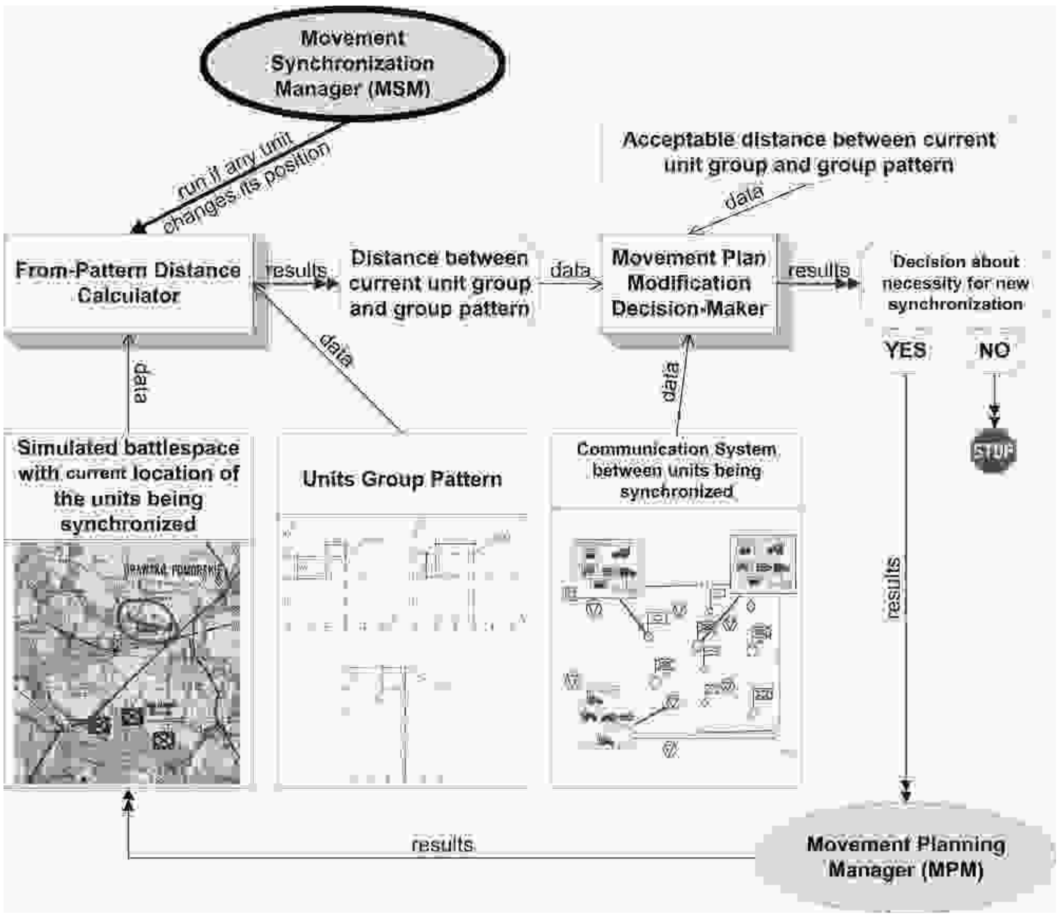

Movement Synchronization Manager (MSM)
run if any unit changes its position
Acceptable distance between current unit group and group pattern
data
From-Pattern Distance Calculator
results
Distance between current unit group and group pattern
data
Movement Plan Modification Decision-Maker
results
Decision about necessity for new synchronization
YES
NO
STOP
data
data
data
Simulated battlespace with current location of the units being synchronized
Units Group Pattern
Communication System between units being synchronized
results
results
Movement Planning Manager (MPM)

Fig. 5.13. The idea of the *Movement Synchronization Manager*

The first step (before simulation) is to run the *Movement Planning Manager* (*MPM*) which plans the movement of *K* objects by solving the optimization problem defined in chapter 5.3.1 (depending on user preferences). The *MSM* is started when the unit movement simulation starts. It keeps information about *group* (*arrangement*) *pattern* (*GP*) of *K* monitored units, type of *distance measure* (*TDM*) between the current group and group pattern, and *acceptable value of distance* (*AVD*). When the simulation starts the *MSM* is informed about each change of location of the monitored units and then the procedure *From-Pattern Distance Calculator* is executed. This procedure calculates the distance from *GP* taking into account the defined distance measure *TDM, AVD* and current locations of *K* units being monitored. Next, the procedure *Movement Plan Modification Decision-Maker* is executed. If a calculated "distance" is greater than the acceptable value of the *AVD* "distance" and communication between the commanding unit and monitored units exists (we simulate a commander, which sees or knows about departures from the plan and decides to synchronize movement of units subordinate to him using the communication network) then the *Movement Planning Manager* (*MPM*) starts to search for a new schedule for the *K* units.

Note that the group pattern units (*GP*) have been defined twofold: (1) using the geographical (terrain) distance; (2) using time. The definition of the time group pattern is presented in chapter 4.2.1.2 as the *MSST* problem (more precisely: as instances of the $\tau_p(k)$ time). The definition of the terrain distance group pattern is presented in chapter 4.2.1.3 as the *MSSD* problem.

5.5. Summary

The models and methods described in the chapter are used in a real simulation support system for military operational training (Antkiewicz *et al.*, 2011b; Najgebauer *et al.*, 2007b) and/or can be used in *Computer Generated Forces* systems. The presented methods and their implementations are very promising in context of *Computer Assisted Exercises* management and effectiveness. By using, for example, a decision automata at the battalion level, we can save a lot of time and decrease the number of training participants, so even very complex exercises can be organized and carried out by analyzing and go through different scenarios of military conflicts.

There are some conclusions related to the presented decision automata. The presented multicriteria weighted graph similarity problem (*MWGSP*) combines well-known structural and rarely considered non-structural (quantitative) similarity between graphs as models of some objects. The approach to structural similarity between graph vertices adopted from (Blondel *et al.*, 2004) can be improved (Melnik *et al.*, 2002; Senellart & Blondel, 2003) because of the definition of the similarity matrix (5.15) is still not totally satisfactory (e.g. it is not always diagonally dominant for self-similarity). Different types of similarities should be compared with graph vertices similarity. Moreover, different types of methods for solving multicriteria problems (Eschenauer *et al.*, 1990) should be checked for solving *MWGSP*. Let us note that we can easily adopt centrality measures from social networks to use them or their combinations instead s_{ij} in (5.17) (Bartosiak *et al.*, 2011).

One of the aspects of automatization of the decision processes – movement planning, synchronization and simulation is essential not only in *CGF* systems. Simulation systems for military trainings should have modules for management (planning, synchronization) multi-objects movement. The quality of this management has an effect on accuracy, effectiveness and other characteristics of simulated battlefield systems. In general, modelling, optimization and simulation of multi-convoy redeployment (for simultaneous movement of many columns) are very complicated processes. Complexity of these processes depends on the following conditions: number of convoys (the greater the number of convoys the more complicated the scheduling of redeployment is); number of objects in each convoy (the longer the convoy the more complicated the scheduling of redeployment is); Have convoys been redeployed simultaneously? Can convoys be destroyed during redeployment? Can the terrain-based network be destroyed during redeployment? Have convoys been redeployed through disjoint routes? Have convoys achieved selected positions (nodes) at a fixed time? Do convoys have to start at the same time? Have convoys determined any action strips for moving? Can convoys be joined and separated during redeployment? Do convoys have to cross through fixed nodes?, etc. Some of these aspects are considered in chapter 5.3.1 and in the papers: (Beautement *et al.*, 2006; Benton *et al.*, 1995; Cassandras *et al.*, 1995; Gelenbe *et al.*, 2004; Karr *et al.*, 1995; Kreitzberg *et al.*, 1990; Lee & Fishwick, 1995; Lee, 1996; Logan & Sloman, 1997; Logan, 1997; Longtin & Megherbi, 1995; Mohn, 1994; Pai & Reissell, 1994; Sahin *et al.*, 2008; Schrijver & Seymour, 1992; Sun *et al.*, 2008; Rajput & Karr, 1994; Tarapata, 1998; 1999b; 2000f; 2001; 2003a; 2004a; 2005a; 2005b; 2007e; 2010b; 2011b; Tuft *et al.*, 2006; Wang, 2006; Wellman *et al.*, 1995; Zafar *et al.*, 2006).

A very important problem, which deals with automatization of decision processes, is the calibration of simulation models of complex processes (Antkiewicz *et al.*, 2006; Dockery & Woodcock, 1993; Hofmann, 2005). It enables the tuning of these models. This process has an influence on one of the most important features of simulation models as is adequacy.

Some additional applications of presented methods are described in chapter 6.

6. Selected Applications in Real Systems

In this chapter some applications in real systems of presented models and algorithms are described. In chapter 6.1 an application and specialization of movement planning and simulation models and algorithms in real simulation systems *Zlocien* and *MSCombat* are presented. Chapter 6.2 contains a description of knowledge-based pattern recognition tools to support mission planning and simulation as an example of tools, which use models and algorithms presented in chapter 5.2. In chapter 6.3 we described applications in security and crisis management systems.

6.1. Movement Planning and Simulation in the *Zlocien* and *MSCombat* Systems

6.1.1. Simulation Based Operational Training Support System (*SBOTSS*) *Zlocien* and *MSCombat*: a Short Overview

The stochastic simulator being considered is the *Simulation Based Operational Training Support System* (*SBOTSS*) – *Zlocien* (Antkiewicz *et al.*, 2008e; 2009b; 2010f; Najgebauer *et al.*, 2004a; 2007b; 2008b; Zlocien, 2002) which has been built at the Cybernetics Faculty of the Military University of Technology in Warsaw (Poland) and the author of this work is a member of the team, which has built the system.

Table 6.1. Description of the *Simulation Based Operational Training Support System* (*SBOTSS*) - *Zlocien*

Feature	*Description*
Domain	Land operations, corps-division-brigade levels. Supported by detailed logistics and integrated intelligence operations, air support, EW.
Span	ADRG digitized maps and *VPF* terrain data permit the model to be used worldwide. The Terrain Rectangle Model (*TRM*) and Road_and_Railway Net Model (*R&RNM*) can be used to build terrain files to support the *Zlocien* model.
Environment	Rectangle-based terrain aggregates regional terrain and environmental characteristics: traffic-ability, elevation, vegetation, chemical contamination, and weather – granularity is 200m×200m. Railways and roads are mapped via the independent Road_and_Railway Net Model, which is complementary to the Terrain Rectangle Model. Specific terrain or engineering objects are modelled separately and can be located on the maps transformed by terrain models – *TRM* and *R&RNM.*
Software	Combat simulator, After Action Review (AAR) procedures, Calibrator, Set of DBs (operational, terrain, scenario), Scenario Editor, Portal *SBOTSS Zlocien*, Reporter AAR, Visualization Server, ADatP3 Editor.

The *SBOTSS Zlocien* has been put into practice at the War Games and Simulation Centre of the National Defence University in Warsaw. This system has been used during *Computer Assisted Exercises* (*CAX*es). The *Zlocien* is an integrated, interactive, multi-sided land, analysis and training support model (with logistics, engineering, electronic warfare and intelligence functions), which realizes stochastic ground-combat attrition. The system is a federation, High Level Architecture (*HLA*) compliant (Kuhl *et al.*, 1999), cooperating with *C*3 systems (*Command, Control and Communication, C3*) and heterogeneous platform (Sun Solaris, Windows NT). The detail description of the *Zlocien* system is presented in Table 6.1. The *TRM* is equivalent to the Z_1 network from (2.2) and the *R&RNM* – to the Z_2 network from (2.4).

The *Modelling and Simulation of Combat* (*MSCombat*) system has been also built at the Cybernetics Faculty of the Military University of Technology in Warsaw. The basic features of the environment are as follows (Najgebauer *et al.*, 1999b): the conflict scenario preparing, mission formulating for two sides, support of decision making process on the division level, simulation of decision making in the lower level, simulation of combat actions (combat units manoeuvre battle simulation), communication simulation, realization of external tasks in the interactive mode, commander interference with game during the simulation process, evaluation of decisions made as a result of data collected which are connected with two fighting sides moves and effects of these moves. The *MSCombat* is realized on the basis of MODSIM III and SIMOBJECT language. The hardware platform is heterogeneous, so simulation can be executed on PC Pentium and Risc platforms. The RTI API enables a co-operation of these platforms. The co-operation RTI API specification environment with MODSIM is possible thanks to special HLA/MODSIM interface. The *General Algebraic Modelling System* (GAMS) supplies methods of optimisation problems solving and is called from the simulation environment.

6.1.2. Models and Algorithms for Movement Planning

Algorithms of movement planning in the *Zlocien* system allow us to determine movement plans defined in chapter 5.3.1.2. The algorithms take into account three types of criteria defined there: time (l_1), distance (l_2) and degree of camouflage (l_3) (or decamouflage). We can find single-criterion paths or multicriteria (2- or 3-criteria) paths taking into account the metacriterion function approach described in chapter 3.3.4.3 with the arc metafunction (3.49).

To find paths for units, modified shortest path algorithms (*SPA*) such as Dijkstra's, *A**, geometric *SPA* are used in *SBOTSS Zlocien* (Tarapata, 2004a; 2011b):

- (A) Dijkstra's for finding shortest paths using binary heaps (with complexity $O(m \log_2 n)$, where m – number of graph edges (arcs), n – number of graph nodes); we can also use faster implementations of the Dijkstra's algorithm, e.g.

using 4-ary heaps (with complexity $O(m \log_4 n)$, see Fig. 6.1), which is very effective for the special structure of the graph (if the graph is r-regular[1] then r-ary heap is very effective to represent priority queue in the Dijkstra's or A* algorithm (Cherkassky *et al.*, 1996; Tarjan, 1983));

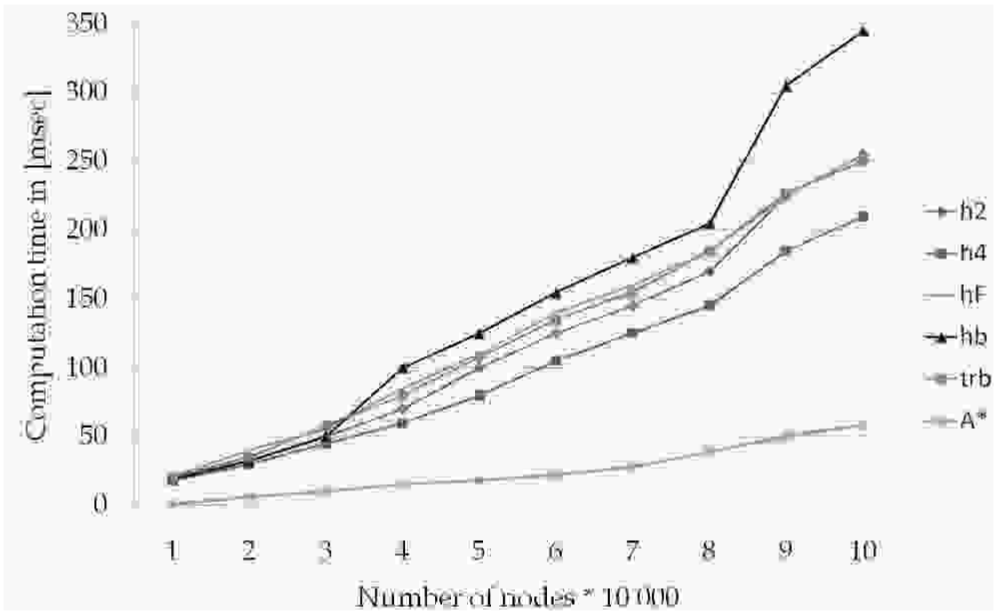


Fig. 6.1. Computational complexity of *A** (A*) and the Dijkstra's algorithms implemented with heaps: binary (h2), 4-ary (h4), Fibonacci's (hF), binomial (hb) and red-black tree (trb) for the real part of terrains used in the *Zlocien* system with four neighbours for each square

- (B) *A** for finding the shortest paths using heuristics (Hart *et al.*, 1968); in the case of grid graphs this algorithm converges faster (in the average case) than the Dijkstra's algorithm. In the *A** algorithm the criterion for choosing node x' for the next iteration is based on the function:

$$g(x') + h(x') = \min\{g(x) + h(x) : x \text{ is not a checked node}\}$$

while, in the Dijkstra's algorithm:

$$g(x') = \min\{g(x) : x \text{ is not a checked node}\}$$

where: $g(x)$ – length of the shortest path from source node s to node x; $h(x)$ – estimation of the length of the shortest path from node x to target node t. It is proved (Hart *et al.*, 1968) that, if the value of the heuristic $h(x)$ is no greater than the real length of the shortest path from node x to target node t, then *A** gives the optimal solution (for $h(x)=0$ we have the Dijkstra's algorithm);

[1] Graph G is r-regular if each of its nodes is adjacent to r nodes.

- (C) for determining the shortest geometric paths (Mitchell, 1999). In the Zlocien system this algorithm supplements two of the above presented algorithms (we obtain the *Hybrid Shortest Path* (*HSP*) algorithm) and it is used in the case when the size of the network S^z is large (the default is 10 000 nodes, but it is a parameter set in the so-called calibrator of the simulation system (Antkiewicz *et al.*, 2006)).

The idea of the hybrid (*HSP*) algorithm is described in Fig. 6.2. First we run *HSP* (C) to determine squares belonging to the segment linking the source square with the target square and next the condition whether all of these squares are passable is checked (starting from the source square). If all of the squares belonging to this segment are passable then the path has been determined. Otherwise, the hybrid algorithm runs one of the algorithms (A) or (B) which start from the last passable square on the segment (or from the one square before the last passable, or from the two squares before the last passable, etc.) and determine the shortest path to the target square. If a path exists then it is joined to the part of the path determined using the *HSP* algorithm (C), otherwise we use (A) or (B) algorithms from the source to the target.

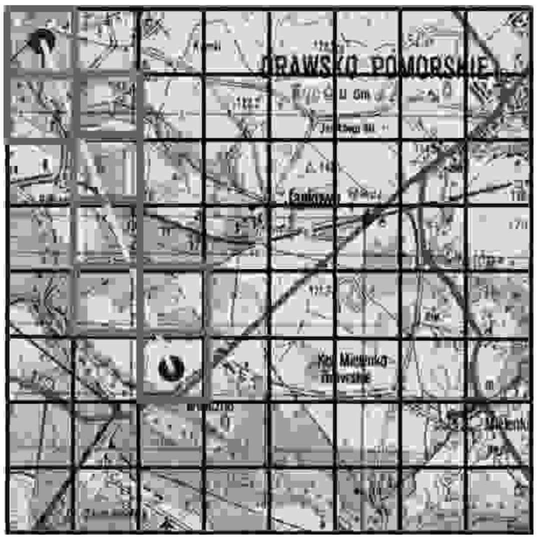


Fig. 6.2. The idea of the hybrid algorithm (*HSP*) for the shortest paths in the *Zlocien* system

Modifications of mentioned algorithms deal with the following details:

- paths determining in different configurations: from point (region) to point (region), visiting selected points (regions), omitting selected points (regions, obstacles), inside or outside selected region, off-roads only, on-roads only, combined on- and off-roads and others to find different types of movement plans defined in chapter 5.3.1.2;

- if we do not set region inside where we want to find the path then the algorithm itself iteratively determines the rectangular region which is based on the line linking source and target points (nodes) of movement, in order to minimize computational time;
- if we want to find the on-road path only, and there are no nodes of the road network (Z_2) inside the intermediate squares, then the algorithm may optionally find crossroads (nodes of the road network) that are nearest to squares inside which the path must cross.

In Table 6.2 the headings of the procedures for finding paths in networks Z_1 and S^z in the *Zlocien* system are presented.

Table 6.2. The headings of the procedures for finding paths in the networks Z_1 (left hand side) and S^z (right hand side)

Procedures for finding paths in Z_1 network	*Procedures for finding paths in S^z network*
procedure *Determine_Path_On_Squares* *(in Unit_Id,* *in Inside_Region,* *in Region_To_Avoid,* *in Region_From,* *in Region_To,* *in Criterion,* *in Whether_Avoid_Occupied,* *out Path);*	procedure *Determine_Path_On_Roads* *(in Unit_Id,* *in Inside_Region,* *in Region_To_Avoid,* *in Across_Region,* *in Criterion,* *in Whether_Search_For_Route_Nodes,* *out Path);*

In the *MSCombat* system the first implementation of the *SGDP* algorithm from chapter 3.4.3.1 for finding K>1 disjoint paths has been tested.

6.1.3. Models and Algorithms for Movement Simulation

Movement simulation of the units is realized in both terrain models Z_1 and Z_2. However, in each of these models we "see" the units during movement in different ways. We accept the following assumptions and definitions:

1. *Small square* and *big square*: small square is the square of the Z_1 network; big square is an aggregated set of small squares (e.g. for the unit at company level the big square is the square with 4x4 small squares);
2. Possible deployment of the units:
   - inside a small square (e.g. reconnaissance patrol on a single vehicle);
   - inside a big square (unit at company level may occupy 4x4 small squares);
   - on the road in a column (based on arcs of Z_2). In this case we "see" the unit on the road twofold: (1) as occupying arcs (part of the roads) and nodes (crossroads) of the Z_2 network, (2) as a sequence of squares of the Z_1 network through which the arc crosses. In this case we use functions $FW_1OnW_2 : W_1 \to 2^{W_2}$ and $FW_2OnW_1 : W_2 \to W_1$ from Table 2.1 and Table 2.2;

3. Inside the big square the unit is evenly deployed taking into account passable small squares only (in special cases we omit this assumption).

Movement of the unit, which is deployed in the big square is being done by determining the sequence of small squares, which create the path for a selected (e.g. lower left) small square of the big square (using the algorithm presented in the previous section) and next we realize the movement from square to square. In such a case we move the big square with the small square granulation (see Fig. 6.3).

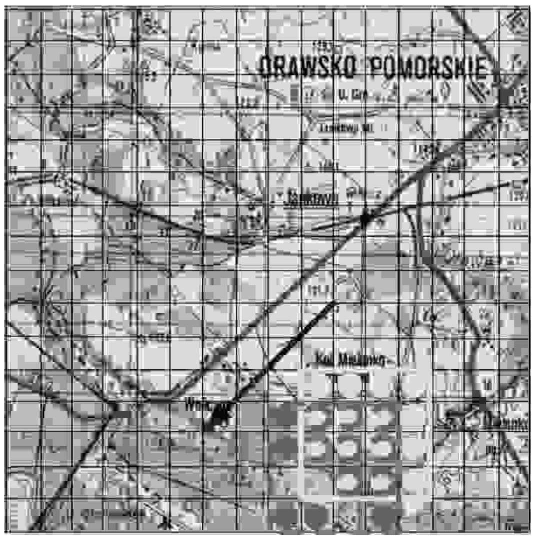


Fig. 6.3. The idea of unit movement on the squares of network Z_1

In Fig. 6.3 the big square in which the unit is deployed has 4x4 small squares. The continuous line describes current deployment of the unit, the dashed line – the new location (after movement with the small square in the south-west direction). Dots inside the squares describes that there is some part of the unit. The arrow from the left side of the big square describes the direction of movement. After the movement the unit has been cumulated inside the 13 small squares because 3 lower left squares are not passable (because of the lake). Movement across the "bottleneck" of the terrain (e.g. minefields crossing, bridge crossing) are realized similarly (using the accumulation of the unit inside small squares).

Movement of the unit on the road (deployed in the column) is done by determining the sequence of nodes (crossroads) and arcs (part of the roads) of the Z_2 network using the algorithm presented in the previous section and next we realize the movement from crossroad to crossroad. As it has been written, we "see" the unit on the road twofold: (1) as occupying arcs (part of the roads) and nodes (crossroads) of the Z_2 network, (2) as sequence of squares of the Z_1 network by which the arc crosses. In the case of (1) we move the head and the tail of the column and we register arcs of the Z_2 on which the head and the tail are located with degrees of crossing these arcs. In the case of (2) we locate the head and the tail of the column on small squares and we move the sequence of small squares (from the head to the tail), like in Fig. 6.2.

In both models of movement the unit can move to the next square of its path if the following criterions are satisfied:

- square is topographically passable;
- square is tactically passable (lack of minefields, lack of an enemy unit (unit can occupy a square of the enemy unit, if and only, if the enemy unit is destroyed), number of own units in the square are no greater than a critical value (default=5)).

The movement can be also interrupted, because of: the lack of fuel, destroying the unit, commander decision, simulation termination, etc. (see description of fault situations in chapter 5.3.2.1).

The very important problem of setting the current velocity of the unit *id* during movement simulation is described in chapter 5.3.2.2.

For movement simulation of units we use simulation procedures similar to these, which have been described in chapter 5.4.

6.1.4. Practical Example

In this chapter a practical example of march planning and simulation in the *Zlocien* system using automata for a march (see chapter 5.3) is presented. In Fig. 6.4 an initial tactical situation is shown.

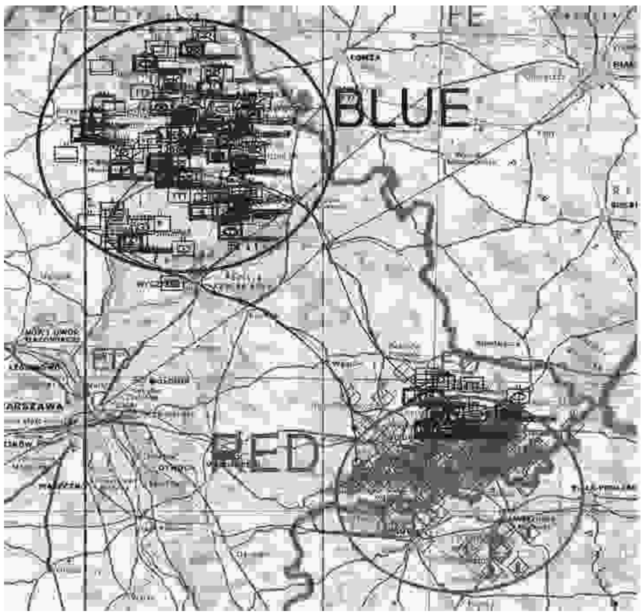


Fig. 6.4. Initial tactical situation, 4:00am: two mechanized brigades of the BLUE conflict side (121 BZ and 123 BZ) receive an order to march

In the example being considered, 2 mechanized brigades (121 BZ and 123 BZ: each of the brigades consists of 4 mechanized battalions × 4 mechanized companies each) of the BLUE side receive the order to march. In the superior order (from (5.31)):

- destination area for 121 BZ and 123 BZ is set about 30km to the north of the northern edge of the location area of the RED conflict side;
- distance from the source area *S* to the destination *D* is equal to about 110km;
- 5 checkpoints are set.

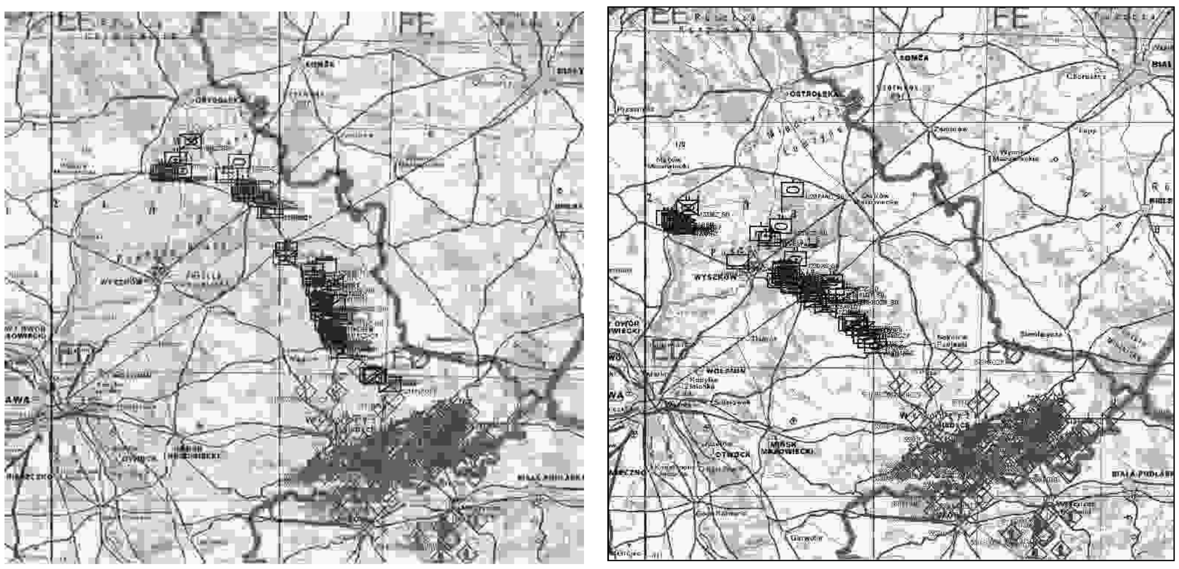

(a) (b)

Fig. 6.5. Location of the 121 BZ (a) and 123 BZ (b) on the road, 5:50am

In Fig. 6.5 the locations of 121 BZ and 123 BZ, respectively, after nearly 2 hours of marching are presented.

Initial redeployment of the BLUE side is presented in Fig. 6.6a. 121 BZ is redeployed on the northern-east of the BLUE force redeployment area. 123 BZ is redeployed south of 121 BZ. In Fig. 6.6b location of 121 BZ and 123 BZ at 5.50am is shown.

In Fig. 6.7 the fuel level percentage regarding the starting level (4 825 litres) is presented for selected unit (12111 kz (belonging to 1211 bz from 121 BZ) consisting of 13 wheeled armoured "Rosomak" carriers) during the 110 km march, from 4:00am to 7:30am. Fuel calculation during a march simulation has been done using formula (5.49).

In Table 6.3 the average velocities between selected march checkpoints (descriptions of *S*, *D*, *PS*, *PD* in chapter 5.3.1.1, see also Fig. 5.10) for 121 BZ and 123 BZ are presented. Average march velocity is equal to about 30km/h. Velocity calculation has been done using procedures described in chapter 5.3.2.2.

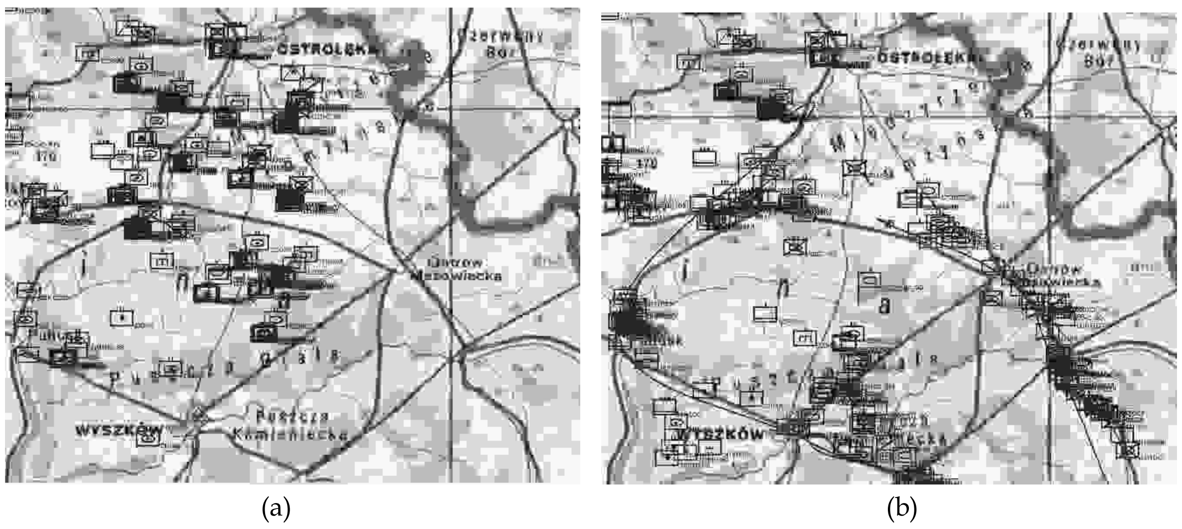


Fig. 6.6. (a) Initial redeployment of the BLUE side, 4:00am and (b) the location of 121 BZ and 123 BZ, 5:50am

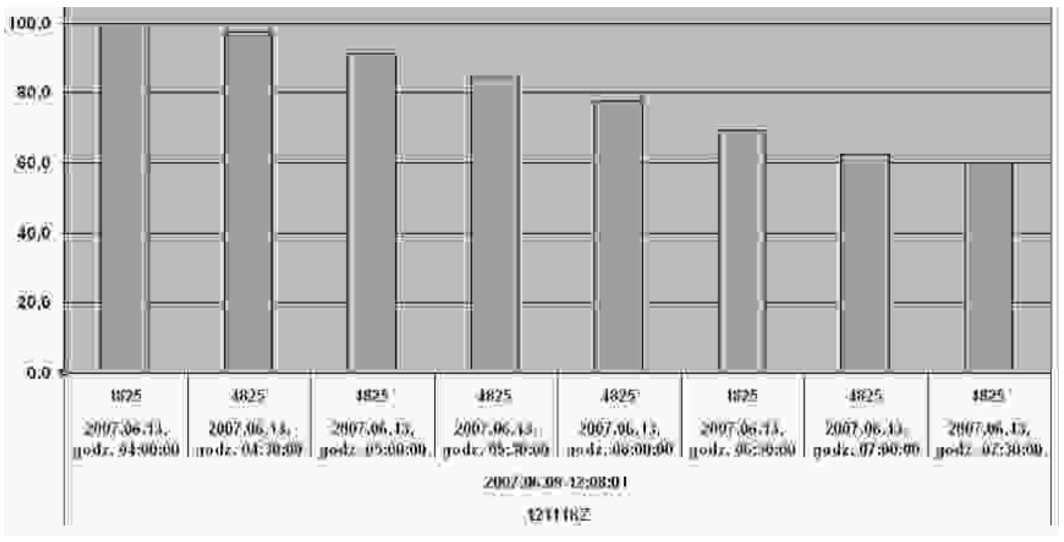


Fig. 6.7. Percentage fuel level regarding the starting level (4 825 litres) for 12111 kz (consisting of 13 wheeled armoured "Rosomak" carriers) during march on the distance 110 km, from 4:00am to 7:30am

Table 6.3. Average velocities between selected march checkpoints for 121 BZ and 123 BZ (in km/h)

Unit	*S=>PS*	*PS=>PD*	*PD=>D*	*S=>D*
121 BZ	12.32	39.65	18.24	29.54
123 BZ	14.07	27.84	22.57	24.65

6.2. Knowledge-Based Pattern Recognition Tools to Support Mission Planning and Simulation

6.2.1. A Short Overview of *CAVaRS* and *Guru* Systems

In this section we present two tools to support mission planning and simulation, which have been built at the Cybernetics Faculty of the Military University of Technology in Warsaw (Poland) and the author of this work is a member of the team which has built them: (1) the deterministic simulator called *CAVaRS* (***C**ourse of **A**ction **V**erification **a**nd **R**ecommendation **S**imulation System*) (Antkiewicz *et al.*, 2011a; 2011b); (2) *The System of Automatic Tools for Decision Support* (*SATDS*) *Guru* (Antkiewicz *et al.*, 2009c; Guru, 2005).

The *CAVaRS* may be used as a part of a bigger system, the *SATDS Guru*, which supports the Polish *C4ISR* systems or it may work standalone. The deterministic and discrete time-driven simulator *CAVaRS* models two-face land conflict of military units on the company/battalion level. The simulator is implemented in the JAVA language. The model concerns a couple of processes of firing interaction and movement executed by a single military unit. These two complementary models use a terrain model described by a network of square areas, which aggregates movement characteristics with 200m×200m granularity (similarly to *Zlocien* system). The course of each process depends on many factors, among them: terrain and weather conditions, conditions and parameters of weapons the units are equipped with, the type of executed unit activities (attack, defence) and the distance between opposite units.

Scenarios of the variants of the military scenario in the Knowledge Base Editor of the simulator *CAVaRS* can be created in two ways: manually and half-automatic. In manual mode the variant can be built using military unit templates stored in the *CAVaRS* database. In half-automatic mode the military scenario can be imported from other *C3(4)ISR* (e.g. *C3ISR Jasmin*) systems using NATO *MIP-DEM* (*Multilateral Interoperability Program - Data Exchange Mechanism*) and NATO *MIP-JC3IEDM* (*Joint Consultation, Command and Control Information Exchange Data Model*) integration database schema. The Knowledge Base Editor can import and transform data from *MIP JC3IEDM* standard data schema to *CAVaRS* data schema. This way is faster than manual mode, because all data of military units or at least most of them can be imported from other *C3(4)ISR* systems with detailed data such as unit location, equipment, weapons, etc.

The purpose of *The System of Automatic Tools for Decision Support* (*SATDS) – Guru* is (Antkiewicz *et al.*, 2009c):

- using expert methods to support decision-making by a commander of an operational (tactical) level concerning planning military actions;

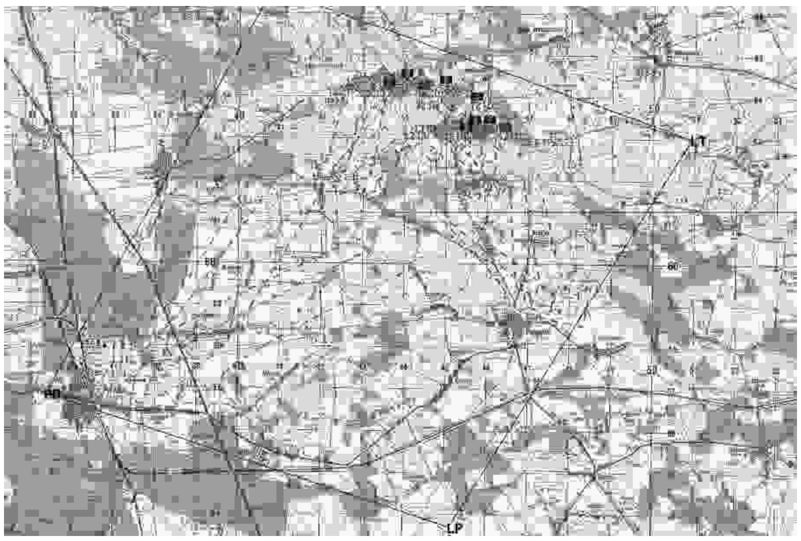
- developing tools for operational training of commanders and field-grade officers in planning military actions;
- provision of software tools for continuing the collection of multiple experts' knowledge and the development of knowledge bases for developing user applications within the scope of expert support for decision-making by appropriate commanders.

The *Guru* is an IT system to support, using expert methods, decision-making in the following Polish C2 (*Command and Control*) systems: for the ground forces – *Kolorado* and *Szafran ZT* (in cooperation with the *Zlocien* system), for the air force – *Dunaj* and *Podbial*, for the navy – *Leba/MCCIS* as well as planning joint operations on the operational level (see also Antkiewicz *et al.*, 2008c; 2010e).

6.2.2. Practical Example of Using ***CAVaRS***

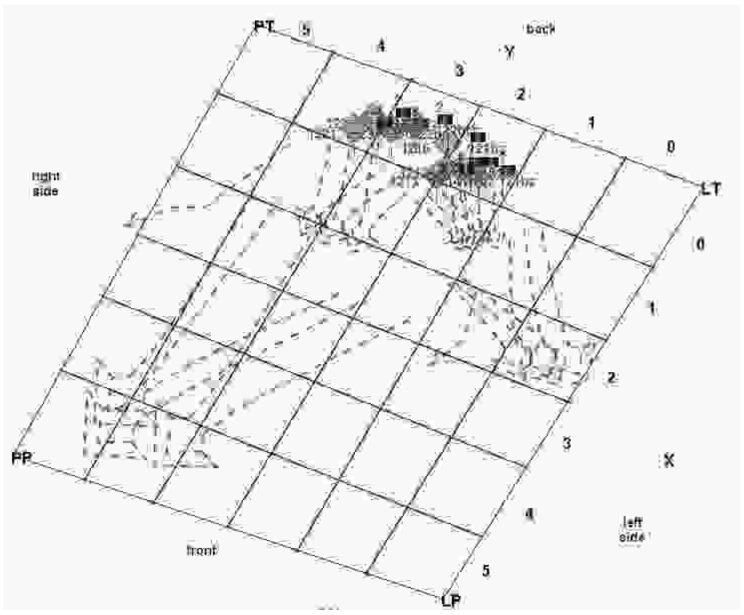
The example shows elements of knowledge base and the algorithm of nearest pattern situation searching based on models defined in chapters 5.2.2 and 5.2.3.

The main element of the system is the knowledge base, which consists of *Pattern Situations* (*PS*) (representation of the *PDSS* set from chapter 5.2.1). Each *PS* is connected to the set of *Course of Actions* (*CoA*). The example of two *PSs* and their *CoAs* are presented in Fig. 6.8.



(a)

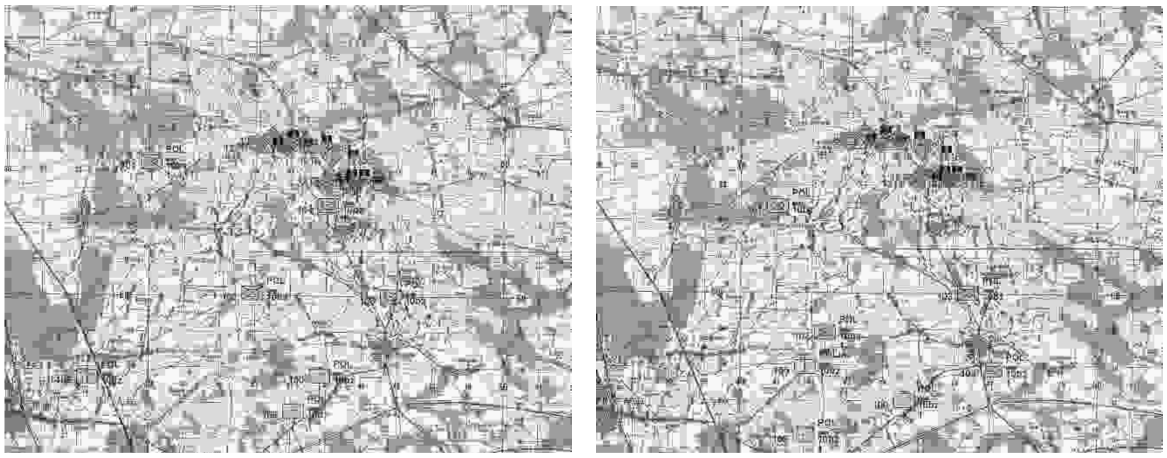
Fig. 6.8. (a) Graphical representation of *PS*$_1$



(b)

Fig. 6.8. (b) An area of opposite forces for PS_1

The first *PS* (PS_1, see Fig. 6.8a) is connected with two *CoA*s (see Fig. 6.9a and Fig. 6.9b). The second PS_2 is shown in Fig. 6.10. Parameters have been fixed for each *PS*. Fig. 6.8b shows the analyzed area of enemy forces. Parameters of each *PS* are kept in the knowledge base (see also Fig. 1.1). Table 6.4 and Table 6.5 show values of *PS* parameters.



(a) (b)

Fig. 6.9. (a) Graphical representation of PS_1, CoA_1; (b) Graphical representation of PS_1, CoA_2

Coordinates of terrain area for PS_1 (NW: north-west corner, NE: north-east corner, SW: south-west corner, SE: south-east corner):

NW (LP)=515556N 0213922E ; NE (PP)=515740N 0213053E ;

SW (LT)=520056N 0214431E ; SE (PT)=520254N 0213541E.

Potential of own forces: mechanized 444; armoured 61.2; artillery 30; antiaircraft 0; other 0.

Table 6.4. Detailed values of PS_1 parameters using notations from (5.1)

i	j	$SD_{ij}^{5,1}$	$SD_{ij}^{5,2}$	$SD_{ij}^{5,3}$	$SD_{ij}^{5,4}$	$SD_{ij}^{5,5}$	$SD_{ij}^{5,6}$	$SD_{ij}^{5,7}$
0	0	54%	1%	1%	0.069	0	0	0
0	1	44%	4%	1%	0.116	0	0	0
0	2	42%	15%	2%	0.186	0	17.46	94.13
0	3	45%	9%	4%	0.21	190	16.32	23.75
0	4	41%	8%	2%	0.252	80	5.2	0
0	5	42%	24%	1%	0.176	0	0	0
1	0	46%	23%	2%	0.12	0	0	0
1	1	54%	5%	1%	0.162	0	0	0
1	2	37%	15%	0%	0.231	0	26.98	140.8
1	3	47%	13%	0%	0.158	25	5.71	21.35
1	4	45%	10%	0%	0.177	25	1.62	0
1	5	35%	0%	34%	0.168	0	0	0
2	0	2%	0%	58%	0.096	0	0	0
2	1	7%	0%	54%	0.135	0	0	0
2	2	17%	0%	50%	0.183	0	0	0
2	3	11%	0%	38%	0.138	0	0	0
2	4	23%	0%	34%	0.162	0	0	0
2	5	51%	0%	29%	0.179	0	0	0
3	0	2%	0%	46%	0.168	0	0	0
...	...	...	...	...	...	...	...	...
5	5	25%	20%	0%	0.013	0	0	0

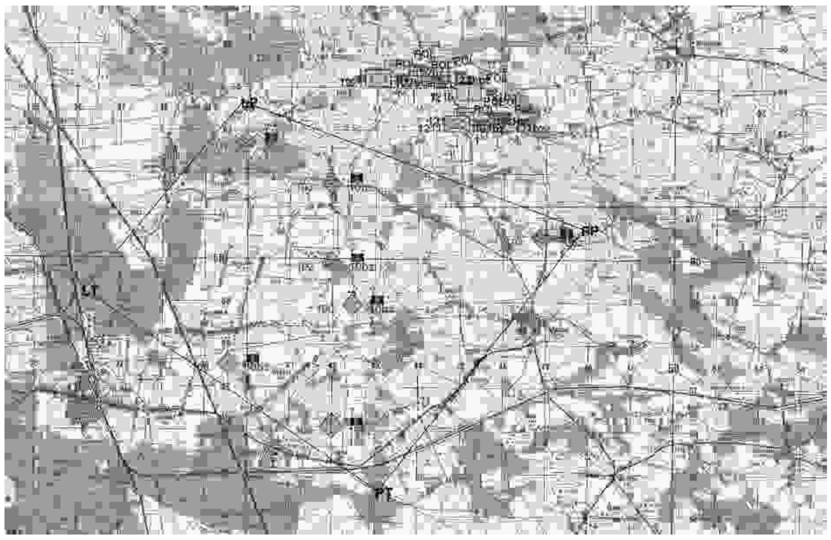


Fig. 6.10. Graphical representation of PS_2

Coordinates of the terrain area for PS_2 (NW: north-west corner, NE: north-east corner, SW: south-west corner, SE: south-east corner):

NW (LP)=520120N 0213451E ; NE (PP)=515943N 0214150E ;

SW (LT)=515858N 0213135E ; SE (PT)=515625N 0213736E.

Potential of own forces: mechanized 320; armoured 73.3; artillery 280; antiaircraft 0; other 0.

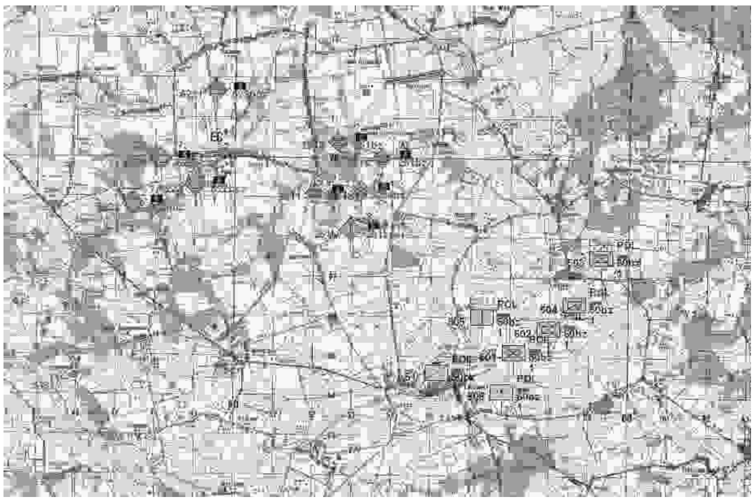


Fig. 6.11. Current situation (*CS*)

Table 6.5. Detailed values of PS_2 parameters using notations from (5.1)

i	j	$SD_{ij}^{5,1}$	$SD_{ij}^{5,2}$	$SD_{ij}^{5,3}$	$SD_{ij}^{5,4}$	$SD_{ij}^{5,5}$	$SD_{ij}^{5,6}$	$SD_{ij}^{5,7}$
0	0	29%	93%	0%	0.01	0	0	0
0	1	55%	48%	0%	0.06	0	0	0
0	2	91%	1%	0%	0.04	8.62	4.49	0
0	3	84%	10%	0%	0.04	5.38	2.81	0
0	4	84%	11%	0%	0.03	0	5.85	27
0	5	76%	30%	0%	0.01	0	0.65	3
...	...	...	...	...	...	...	...	...
2	2	88%	0%	0%	0.03	13	1.44	0
2	3	84%	10%	0%	0.05	60	6.55	0
2	4	59%	44%	0%	0.07	6	0.6	0
2	5	77%	12%	0%	0.06	0	0	0
3	0	66%	33%	0%	0.09	0	0	0
3	1	83%	4%	0%	0.04	0	0	0
3	2	88%	3%	0%	0.02	6.5	0.72	0
3	3	80%	7%	0%	0.08	32.5	3.59	0
3	4	82%	1%	0%	0.1	0	0	0
3	5	81%	0%	0%	0.12	0	0	0
4	0	40%	74%	0%	0.08	66.9	7.39	0
4	1	62%	43%	0%	0.06	32.7	3.61	0
4	2	85%	1%	0%	0.05	93.6	10.4	0
4	3	70%	22%	0%	0.09	0	0	0
4	4	69%	9%	0%	0.15	0	0	0
4	5	87%	4%	0%	0.05	18.9	2.09	0
...	...	...	...	...	...	...	...	...
5	5	85%	6%	0%	0.05	85.1	9.41	0

Values of each *PS* parameters of the current situation (see Fig. 6.11) have been calculated. The algorithm for finding the most similar pattern situation compares the current situation parameters with each *PS* from the knowledge base using the method described in chapters 5.2.2 (the method from chapter 5.2.3 is still being developed and tested). As a result, the PS_1 has been fixed according to the *dist* values (equations (5.7) and (5.9)) presented in Table 6.6 because:

$$dist_{pot}(CS,PS_1) < dist_{pot}(CS,PS_2) \text{ and } dist_{ter}(CS,PS_1) < dist_{ter}(CS,PS_2),$$

hence PS_1 dominates PS_2 from the R_D (formula (5.12)) point of view.

Table 6.6. Detailed values of *dist* parameters from (5.7) and (5.9)

PS	$dist_{pot}(CS,PS)$	$dist_{ter}(CS,PS)$
PS_1	203.61	1.22
PS_2	222.32	1.47

6.3. Applications in Security and Crisis Management Systems

6.3.1. *MWGSP* Approach

In this chapter we will show how to use, described in chapter 5.2.3, the *MWGSP* approach with the *Social Network* (represented by *Complex Network*) analyzing and semantic-based terrorist threat indication as well as information diffusion in networks. Applications presented here are described in detail in (Bartosiak *et al.*, 2011; Tarapata & Kasprzyk, 2009c; 2010e; Tarapata *et al.*, 2010d).

The method presented in (Tarapata *et al.*, 2010d) introduces an original approach to knowledge representation as a semantic model, which is further processed by the inference algorithms and structure graph analysis towards a complex network (using the *MWGSP* model). Described models consist of experience gathered from intelligence experts and several open Internet knowledge systems such as the *Global Terrorism Database*. We have managed to extract core information from several ontologies and fuse them into one domain model aimed at providing the basis for indirect associations' identification method. Until now, scientists have tried to construct theoretical models describing the behaviour of real systems, which is the main reason of *Complex Networks* applications (Antkiewicz *et al.*, 2009a; Barabasi & Reka, 2002; Kasprzyk, 2005; Newman, 2003; Strogatz, 2001; Watts & Strogatz, 1998). The main aim of research in this area is to uncover the mechanisms hidden in the structure of complex systems, which can further lead to the discovery of real networks characteristics and their explanation. Apparently, networks derived from real data (most often spontaneously growing) have "six degree of separation", power low degree

distributions, hubs occurring and many other interesting features. *Complex Networks* have *Scale Free*, *Small Word* and *Clustering* properties (Wang & Chen, 2003) that make them accurate models of networks such as social networks, in particular, a terrorist organization with features mentioned above (Antkiewicz *et al.*, 2008b; Najgebauer *et al.*, 2007a). The *Scale Free* and *Small World* networks, while being fault tolerant, are still very prone to acts of terrorism. The *Scale Free* feature distinguish immunity against random attacks (it is hard to hit a hub). The *Small World* feature can dramatically affect communication among network nodes.
For the purpose of this application we have developed a transformation from a created semantic network into a set of *Complex Networks*. First we have to choose the ontology, which is the most significant from the analysis point of view. It leads to leave only a subset of nodes and edges connecting them. At this moment we have produced a graph with a uniform node and edge type. As a result of the transformation, one of the possible *Complex Networks* has been generated. In order to find a representation of a terrorist organization as a complex network, we should apply the algorithm presented in Fig. 6.12.

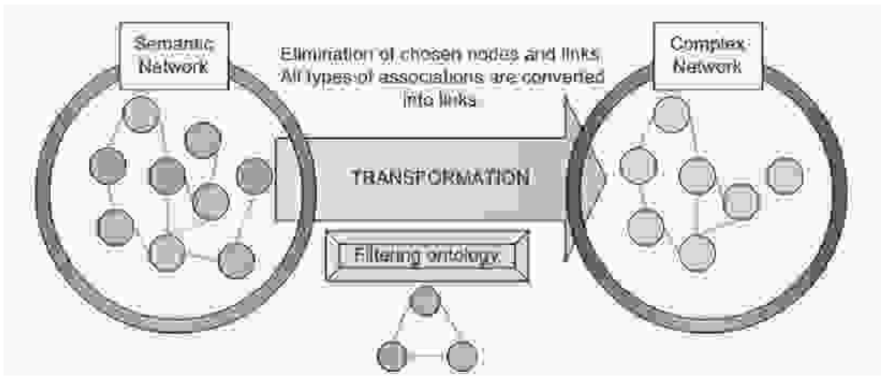

Semantic Network
Elimination of chosen nodes and links
All types of associations are converted into links
Complex Network
TRANSFORMATION
Filtering ontology

Fig. 6.12. The transition between a semantic network and a complex network using ontology filtering (Tarapata *et al.*, 2010d)

Formally, we can write transformation T of the semantic network S_1 into a complex network S_2 as follows (Tarapata *et. al*, 2010d):

$T: S_1 \xrightarrow{FO} S_2$, where FO describes filtering ontology from Fig. 6.12, and: $S_1 = \left\langle G_1 = \langle N_1, A_1 \rangle, \{f_{i1}(n)\}_{\substack{i\in\{1,\dots,LF_1\} \\ n\in N_1}}, \{h_{j1}(a)\}_{\substack{j\in\{1,\dots,LH_1\} \\ a\in A_1}} \right\rangle$, N_1, A_1 – sets of graph nodes and arcs, respectively, $f_{i1}: N_1 \to Z_{i1}$ – the i-th function described on the graph nodes, $i = 1,\dots LF_1$, (LF_1 – number of node's functions); $h_{j1}: A_1 \to Z_{j1}$ – the j-th function described on the graph arcs, $j = 1,\dots LH_1$ (LH_1–number of arc functions), Z_* – any set (e.g. types of vertices); S_2 – defined by analogy but it has a single function described on the nodes and arcs:

$$S_2 = \left\langle G_2 = \langle N_2, A_2 \rangle, \{f_2(n)\}_{n\in N_2}, \{h_2(a)\}_{a\in A_2} \right\rangle$$

Next, the *MWGSP* approach may be used to analyse such a knowledge representation. For example, in Fig. 6.13 we have a terrorist net that was prepared and executed during the September 11, 2001 attacks (Krebs, 2000) and in Fig. 6.14 we have a subnet of a terrorist net that hijacked airplanes on the September 11, 2001 with two cases: a long time before hijacking (a) and a short time before hijacking (b).

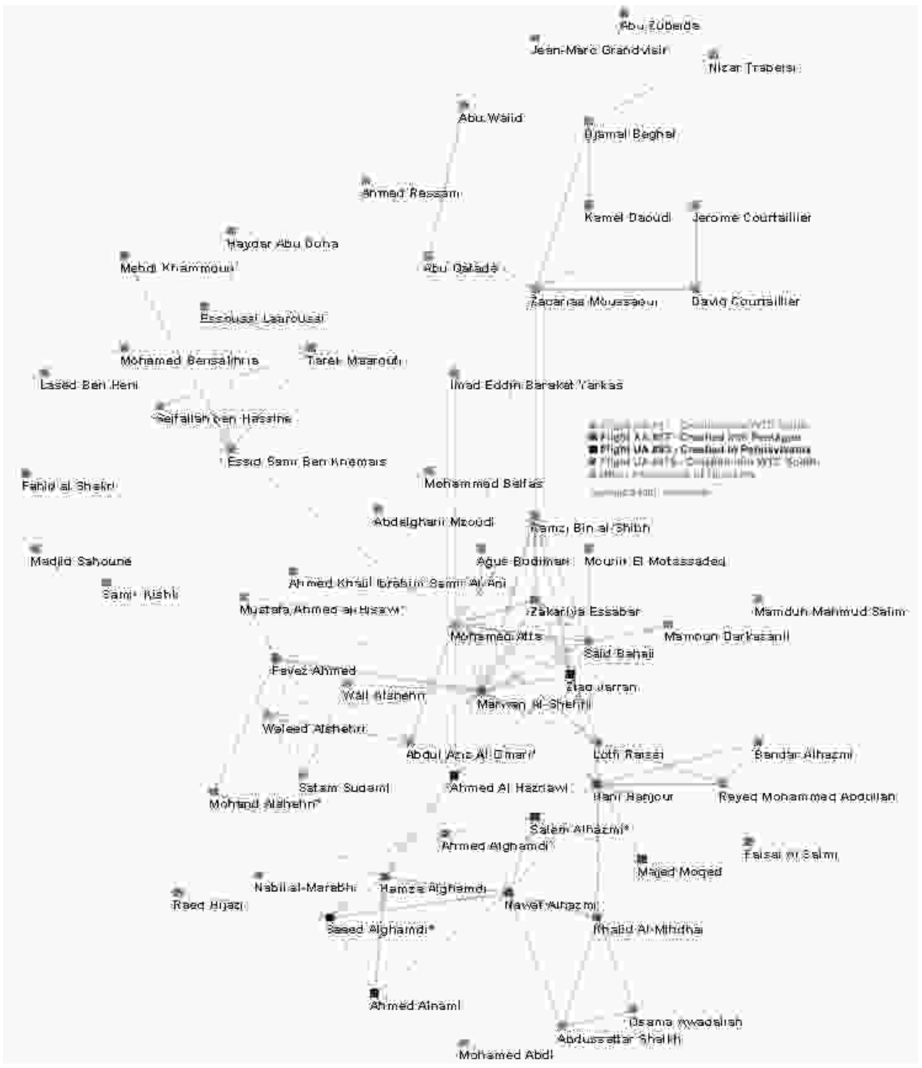
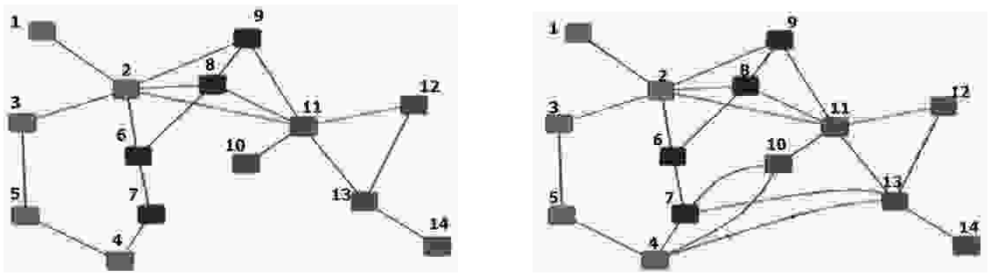


Fig. 6.13. Terrorist net that prepared and executed the September 11, 2001 attacks (Krebs, 2000)

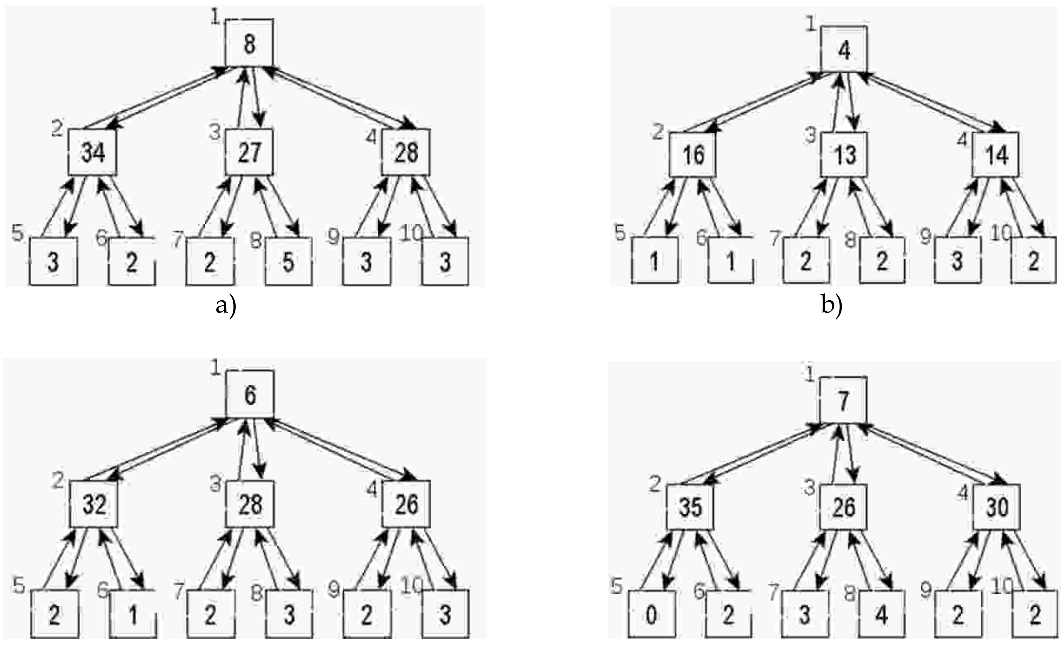
When we use the network from Fig. 6.14 as a "normal" communication between terrorists then we can use the *MWGSP* approach to recognize the threat situation (structural similarity between the net from Fig. 6.14a and Fig. 6.14a (self-similarity) is equal 0.990 and between the net from Fig. 6.14a and Fig. 6.14b is equal 0.880: this difference can indicate a threat situation; we can also set a threshold value for similarity changes, which in the opinion of experts, indicate a threat situation). We can also use quantitative description of these networks (similar to Fig. 6.15) and analyse them using *MWGSP* approach.



a) b)

Fig. 6.14. Terrorist net that hijacked airplanes on the September 11, 2001: (a) a long time before hijacking; (b) a short time before hijacking

Another application is presented in Fig. 6.15. Here we have shown a communication network (e-mails) in a company with a hierarchical structure (we can consider, for example, a communication network inside a criminal/terrorist organization by analogy). Nodes represent users/workers of the company. The number inside each node describes the number of e-mails that were sent by users. In Fig. 6.15a the e-mails net for a "normal" week has been presented. In Fig. 6.15b, Fig. 6.15c, Fig. 6.15d communications inside the company have been shown for every week (for the fixed time window the length equals 1 week). Let us observe that the 1st week is at least a similar week to a "normal" week (see Table 6.7: for the 1st week we obtain the smallest value of the scalar function $H(G)$ from (5.27), which combines a structural and a quantitative similarity to a "normal" week).


a) 1: 8; 2: 34; 3: 27; 4: 28; 5: 3; 6: 2; 7: 2; 8: 5; 9: 3; 10: 3
b) 1: 4; 2: 16; 3: 13; 4: 14; 5: 1; 6: 1; 7: 2; 8: 2; 9: 3; 10: 2
c) 1: 6; 2: 32; 3: 28; 4: 26; 5: 2; 6: 1; 7: 2; 8: 3; 9: 2; 10: 3
d) 1: 7; 2: 35; 3: 26; 4: 30; 5: 0; 6: 2; 7: 3; 8: 4; 9: 2; 10: 2

a) b) c) d)

Fig. 6.15. E-mails net in hierarchical company: (a) "normal" week; (b) 1st week; (c) 2nd week; (d) 3rd week

Table 6.7. Values of scalar function $H(G)$ combining structural (weight α_1) and quantitative (weight α_2) similarity measures between graphs representing the e-mail net during a "normal" week and between 3 weeks from Fig. 6.15

Compared graphs	*Weights* (α_1; α_2)		
	(0; 1)	(0.5; 0.5)	(1; 0)
1st week net	-0.694	0.144	1
2nd week net	-0.646	**0.177**	1
3rd week net	**-0.643**	0.160	1

The presented *MWGSP* idea is an original attempt at integrating theories and practices from many areas, in particular: semantic models, social networks, graph and network theory, decision theory, data mining and security, as well as multicriteria optimization. It utilizes the theoretical basis for a very practical purpose of growing importance and demand: widely understood countering terrorism. Moreover, the presented approach combines well-known structural and rarely considered non-structural (quantitative) similarity between graphs as models of objects and can significantly improve social network analysis. Our models and methods of network analysis have been used in the criminal justice domain to search large datasets for associations between crime entities in order to facilitate crime investigation. Let us note that we can easily adopt centrality measures from social networks to use them or their combinations instead s_{ij} in (5.17) (Bartosiak *et al.*, 2011; Tarapata *et al.*, 2010d). It is also possible to use *MWGSP* approach in medical applications to recognize of illness patterns (Ameljańczyk, 2010a; 2010b). However indirect association and link analysis still faces many challenging problems, such as information overload, high search complexity, and heavy reliance on domain knowledge. In the papers (Antkiewicz *et al.*, 2009a; Kasprzyk, 2008; Tarapata & Kasprzyk, 2010e) have been shown why and how the *Complex Networks* with the *Scale Free* and *Small World* feature can help optimize the topology of communication networks. The first term – *Scale Free* feature – is a good protection against random attacks (it is hard to hit a central node). The second term – *Small World* feature – can dramatically affect communication among network nodes. Thus both concepts and underlying theories are highly pertaining to the presenting idea subject and objectives.

6.3.2. Specific Paths Planning Models

Specific paths planning models applied to crisis management systems, paths planning for transport of hazardous materials have been described in other works of the author (Tarapata, 1999c; 2000e; 2006b; 2008f; Tarapata & Daleki, 2008g; Tarapata *et al.*, 2009b; 2009d; Tarapata & Mierzejewski, 2010f).

One of the most important models being used for paths planning in many applications (e.g. in crisis management systems) are *time-dependent networks* (*TDN*)

(Brodal & Jacob, 2004; Cooke & Halsey, 1996; Dean, 2004; Kaufman *et al.*, 1993; Orda & Rom, 1990; 1991; 1996; Sherali *et al.*, 1998; Tarapata, 1999c; 2000e; 2008f; Wellman *et al.*, 1995; Wu *et al.*, 2005). *TDN* is the network in which at least one function (described on the arcs and/or on the nodes) depends on time. For example, in a road traffic network travel time between two crossroads *i* and *k* (nodes of the network) depend on the starting time in the *i*-th crossroad. This time depends on: traffic lights configuration, traffic jams, current road load (different values in peak hours and outside peak hours), etc. Let us note that network $S_o(t)$ defined in chapter 2.3 is time-dependent, too.

In generality, a time-dependent network $S(t)$ can be defined as follows:

$$S(t) = \langle G, \varnothing, D(t) \rangle \tag{6.1}$$

where: $G = (V_G, A_G)$ – Berge's graph (sometimes, we define $G(t)$ instead G for dynamically changed structure of the network), V_G – set of G nodes (e.g. crossroads), $A_G \subset V_G \times V_G$ – set of G arcs (links) (e.g. road parts between crossroads), $\overline{\overline{A_G}} = A$, $\overline{\overline{V_G}} = V$, $D(t) = \{d_{ik}(t) : (i,k) \in A_G\}$ – set of delay functions, $d_{ik}(t)$ – delay function between nodes *i* and *k*, $\underset{(i,k)\in A_G}{\forall} d_{ik} : T \to T$, *T* – set of moments, $T=[0,\infty)$. The function $d_{ik}(t)$ describes the time value, which is needed to travel between nodes *i* and *k* taking into account that the starting moment from node *i* is equal to *t*.

We define two types of link (arc) models (Orda & Rom, 1990):

- *frozen link model* – time and cost of travel by a link is constant, i.e. when we start from *i* to *k* in the moment of t_0, we need exactly a time of $d_{ik}(t_0)$ and we can achieve the *k* node in the moment $t_1 = t_0 + d_{ik}(t_0)$;
- *elastic link model* – time cost of travel by a link is changeable, i.e. when we start from *i* to *k* in the moment of t_0 we will achieve the *k* node in such a first moment of $t_1 \geq t_0$, that the following formula is fulfilled: $t_1 - t_0 \geq d_{ik}(t_1)$.

Additionally, two different types of link models are considered (Orda & Rom, 1990):

- *FIFO link model* – for arc (i,k) the following formula is satisfied:

$$\underset{t_1 > t_0}{\forall} t_1 + d_{ik}(t_1) \geq t_0 + d_{ik}(t_0) \tag{6.2}$$

- *non-FIFO link model* – for arc (i,k) the following formula is satisfied (see Fig. 6.16):

$$\underset{t_1 > t_0}{\exists} t_1 + d_{ik}(t_1) < t_0 + d_{ik}(t_0) \tag{6.3}$$

It is important to note, that the *elastic link* is always a *FIFO link,* but the *frozen link* cannot be a *FIFO link*. If all arcs of $S(t)$ are *FIFO links*, then network $S(t)$ is a *FIFO* *network*. Otherwise we have a *non-FIFO network*. Most often in practice we have *FIFO networks*. In *non-FIFO networks* we can start later and achieve the destination node earlier than usually (see Fig. 6.16).

From the *non-FIFO* property result three different politics of travel across the network (Orda & Rom, 1990; 1991):

- *unrestricted waiting* – unrestricted waiting is permitted in all nodes of the network (like in the public transportation network: e.g. bus-stops can be a "depository" where a traveller can wait for a good bus);
- *forbidden waiting* – waiting is prohibited in all nodes of the network;
- *source waiting* – waiting is prohibited in all nodes of the network excluding the source node.

For the case in which the waiting is permitted, the function $D_{ik}(t)$ describing the travel time from node i to node k with an optimal waiting time in node i is defined as follows (see Fig. 6.16):

$$D_{ik}(t) = \min_{\tau \geq 0}\{\tau + d_{ik}(t+\tau)\} \tag{6.4}$$

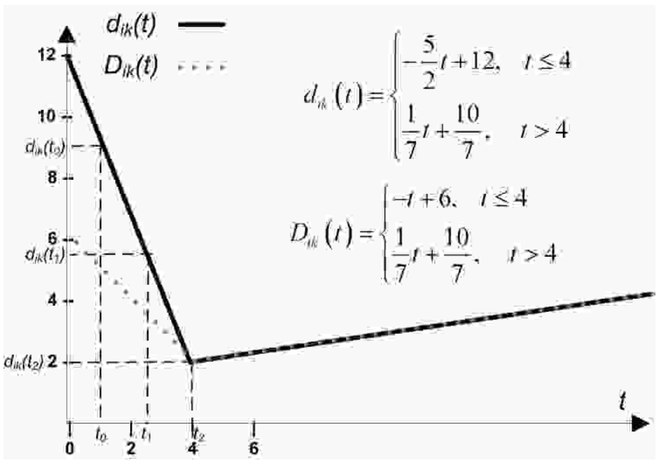


Fig. 6.16. Graph of the function $d_{ik}(t)$ and related to this the function $D_{ik}(t)$ (see (6.4)). Function $d_{ik}(t)$ describes the *non-FIFO* arc, because, for example, for $t_2 > t_0$: $t_2 + d_{ik}(t_2) < t_0 + d_{ik}(t_0)$ $(4 + 2 < 1 + 9,5)$

In the paper (Tarapata, 2006a) a routing problem in computer networks (similar to path planning in road networks) with a *non-FIFO* model of the network as well as permitted and prohibited waiting in nodes (two different cases) has been considered. The modified Dijkstra's algorithm with the arc function $d_{ik}(t)$ (or in the second case $D_{ik}(t)$) based on the approach presented in (Orda & Rom, 1990) has been used. Functions $d_{ik}(t)$ or $D_{ik}(t)$ have forms, which allow accommodating to the predicted network load (capacity) by using the network load prediction model with piecewise linear function based on historical load (see details in (Tarapata, 2006a)). Computational complexity of the algorithm is the same as the Dijkstra's (with the assumption that we have the $d_{ik}(t)$ function) that is $O(A\log V)$. Practical examples of using the method in transportation threat prediction, identification and countering have been considered in details in (Tarapata, 2006c; 2007c; Tarapata *et al.*, 2009b; 2009d; Tarapata & Mierzejewski, 2010f).

Another interesting traffic model, which can be used to model transportation threats (e.g. transport of hazardous materials) and borrowed from car navigation systems is the *simplest path model* (Duckham & Kulik, 2003; Mark, 1986). The simplest path algorithm does not use distance or any other metric information. The algorithm computes the simplest paths using only a measure of *intersection (navigation instruction) complexity* proposed in the work (Mark, 1986). Intersection complexity is classified into *frames*, each frame having several *slots* for different elements of an intersection. A generic turn intersection is modelled as a frame containing a total of 9 slots. Each slot covers information on whether to turn left or right (3 slots), how to recognize the moment to turn (2 slots), how to recognize if the navigator has gone too far (1 slot), and a summary information providing an overview of the turn (3 slots). We can note that the greater value of the measure the more complicated the description of the intersection; hence the *simplest path* is such that the total value of the intersection complexity measure for all nodes belonging to a path is minimal among other paths. This measure can be treated as a kind of penalty for intersection crossing. In context of transportation threats this measure may describe a manoeuvre risk on the intersection. In particular, if we plan a transport of hazardous materials then one of the minimization criteria may be the total manoeuvre risks on a path. Therefore, in this case we may find the simplest path for which the total manoeuvre risk is minimal. Authors of the paper (Duckham & Kulik, 2003) have proposed some modifications of the approach presented by Mark. In the paper (Tarapata *et al.*, 2009d) it has been shown how to use the idea of simplest paths to plan hazardous materials transportation.

One of the methods to find the simplest path is the definition of transformation μ from graph $G=(V,E)$ to graph $G'=(E',\mathcal{E})$. The set of nodes E' in G' is created from the set of arcs E in G where direction of arcs are ignored (i.e. $(v_i,v_j)=(v_j,v_i)$ in E'), but $((u_1,u_2),(v_1,v_2))\in\mathcal{E}$, if both v_1 and v_2 are achievable from u_1 and u_2 in G (see Fig. 6.17). Arc cost function c will have an interpretation of the intersection complexity measure. To find the simplest path from v in G we must find the shortest path from any node of G', which contains v, using the arc function c.

The presented algorithm can be considered in two stages: (1) transformation from G to G' and (2) finding the shortest paths from the source node to the destination using one of the shortest paths algorithms, e.g. the Dijkstra's algorithm. Complexity of this algorithm is equal $O(mn+m'\log n')$, where n, n' – number of nodes in G and G', m, m' – number of arcs in G and G', respectively.

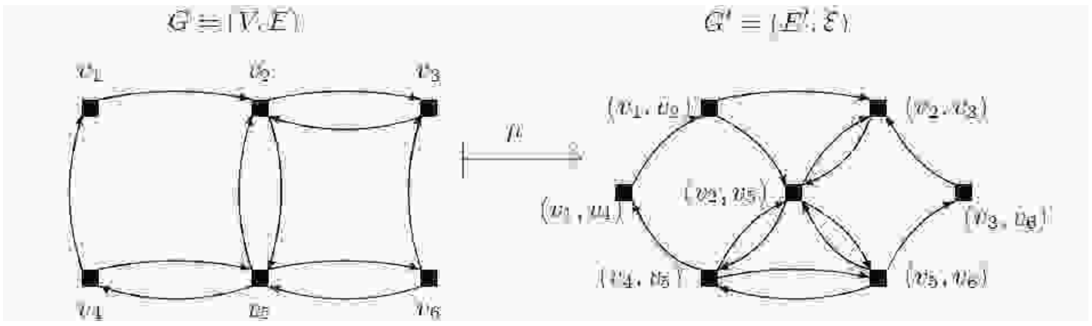


Fig. 6.17. Transformation μ of G in G' (Duckham & Kulik, 2003)

To model the problem of hazardous (e.g. chemical) materials transportation the disjoint path planning models are also used to minimize risk of a potential accident and chemical threats. Then, we can use algorithms presented in chapter 3.4 to solve the problem. The other important problems related to hazardous materials transportation and algorithms to solve them are presented in the works (Berman *et al.*, 2007; Bianco *et al.*, 2009; Carotenuto *et al.*, 2007a; 2007b; Chen *et al.*, 2008; Cox, 1984; Erkut *et al.*, 2003; Wijeratne *et al.*, 1993).

Presented models and algorithms can be also used in selected problems for modelling and optimization of transportation systems (Ambroziak, 1998; Jacyna, 2009).

Bibliography

Aggarwal, A., Kleinberg, J., Williamson, D. (2000): Node-Disjoint Paths on the Mesh and a New Trade-Off in VLSI Layout, *SIAM Journal on Computing*, vol. 29(4), pp. 1321-1333.

Ahuja R.K., Magnanti T.L., Orlin J.B. (1993): Network Flows: Theory, Algorithms and Applications, *Prentice Hall*, Englewood Cliffs.

Ahuja R.K., Mehlhorn K., Orlin J.B., Tarjan R.E. (1988): Faster Algorithms for the Shortest Path Problem, Technical Report 193, *MIT Operations Research Center*.

Almohamad H., Duffuaa S. (1993): A Linear Programming Approach for the Weighted Graph Matching Problem, *IEEE Trans. Pattern Anal. Mach. Intelligence*, vol. 15(5), pp. 522–525.

Ambroziak T. (1998): Modelowanie procesów technologicznych w transporcie, *Oficyna Wydawnicza Politechniki Warszawskiej*, Warszawa.

Ameljańczyk A. (1984): *Optymalizacja wielokryterialna w problemach zarządzania i sterowania*, Ossolineum, Kraków.

Ameljańczyk A. (2010a): Wielokryterialne mechanizmy wspomagania podejmowania decyzji klinicznych w modelu repozytorium w oparciu o wzorce, *Biuletyn Instytutu Systemów Informatycznych*, No. 5, pp. 1-6.

Ameljańczyk A. (2010b): Model informatycznego modułu wspomagania decyzyjnego ustalania wstępnej diagnozy medycznej, *Biuletyn Instytutu Systemów Informatycznych*, No. 5, pp. 7-12.

Andersen, R., Chung, F., Sen, A., Xue , G. (2004): On Disjoint Path Pairs with Wavelength Continuity Constraint in WDM Networks, *Proceedings of the IEEE Infocom'2004*, Hong Kong (China), March 7-11, pp. 524-535.

Antkiewicz R., Manikowski A., Tarapata Z. (2000): Analiza aktualnych i identyfikacja przyszłych wojskowych zastosowań badań operacyjnych, *Sprawozdanie końcowe z PBW 907/99*, Wydział Cybernetyki, Wojskowa Akademia Techniczna, Warszawa.

Antkiewicz R., Najgebauer A., Tarapata Z. (2003): The Decision Automata for Command and Control Simulation on the Tactical Level, *Proceedings of The 5th NATO Regional Conference on Military Communication and Information Systems*, October 7-10, Zegrze (Poland), (CD publication).

Antkiewicz R., Najgebauer A., Rulka J., Tarapata Z. (2004a): Koncepcje i implementacje automatu decyzyjnego w systemie symulacyjnego wspomagania szkolenia operacyjnego, *Materiały XII Konferencji Naukowej „Automatyzacja Dowodzenia"*, 2-4 czerwca, Gdynia-Jurata, (publikacja na CD).

Antkiewicz R., Najgebauer A., Tarapata Z. (2004b): Automat decyzyjny dla potrzeb symulacji dowodzenia na szczeblu taktycznym, *Materiały Konferencyjne z X Warsztatów PTSK*, Kraków, ISBN 83-7242-318-0, pp. 9-16.

Antkiewicz R., Najgebauer A., Rulka J., Tarapata Z. (2006): Calibration of Simulation Models of Selected Battlefield Processes, *Proceedings of the 1st Military Communication and Information Systems Conference*, ISBN 83-920120-1-1, 18-19 września, Gdynia (Poland).

Antkiewicz R., Chmielewski M., Kasprzyk R., Koszela J., Kulas W., Najgebauer A., Pierzchała D., Rulka J., Tarapata Z., Wantoch-Rekowski R. (2008a): Systemy wspomagania zarządzania kryzysowego, w: Kasprzyk J., Najgebauer A., Sienkiewicz P. (red), *Badania operacyjne i systemowe a zagadnienia społeczeństwa informacyjnego, bezpieczeństwa i walki*, ISBN 83-894-7518-9, PAN IBS, Warszawa, pp. 123-136.

Antkiewicz R., Chmielewski M., Kasprzyk R., Kulas W., Najgebauer A., Pierzchała D., Rulka J., Tarapata Z. (2008b): Metody predykcji zagrożenia atakiem terrorystycznym, w: Kasprzyk J., Najgebauer A., Sienkiewicz P. (red), *Badania operacyjne i systemowe a zagadnienia społeczeństwa informacyjnego, bezpieczeństwa i walki*, ISBN 83-894-7518-9, PAN IBS, Warszawa, pp. 147-166.

Antkiewicz R., Gąsecki A., Najgebauer A., Pierzchała D., Tarapata Z. (2008c): Computer Support for Joint Operation Planning Processes, *Proceedings of the Military Communications and Information Systems Conference MCC'2008*, ISBN 83-920120-5-4, September 23-24, Cracow, Poland.

Antkiewicz R., Kulas W., Najgebauer A., Pierzchała D., Rulka J., Tarapata Z., Wantoch-Rekowski R. (2008d): Modelling and Simulation of C2 Processes Based on Cases in the Operational Simulation System for CAXes, *Biuletyn Wojskowej Akademii Technicznej*, 4(652), vol. LVII, pp. 9-24.

Antkiewicz R., Kulas W., Najgebauer A., Pierzchała D., Rulka J., Tarapata Z., Wantoch-Rekowski R. (2008e): Some Aspects of Designing and Using Deterministic and Stochastic Simulators for Military Trainings and CAX'es, *Proceedings of the Military Communications and Information Systems Conference MCC'2008,* ISBN 83--920120-5-4, September 23-24, Cracow, Poland.

Antkiewicz R., Kasprzyk R., Najgebauer A., Tarapata Z. (2009a): The Concept of C4IS Topology Optimization Using Complex Network, *Proceedings of the Military Communications and Information Systems Conference MCC'2009*, ISBN 978-80-7231--678-6, September 29-30, Prague, Czech Republic.

Antkiewicz R., Kulas W., Najgebauer A., Pierzchała D., Rulka J., Tarapata Z., Wantoch-Rekowski R. (2009b): Wybrane problemy projektowania i użytkowania symulatorów walki, *Materiały konferencyjne XV Warsztatów Naukowych PTSK*, Zakopane, 25-27 września, Warszawa , pp. 7-18.

Antkiewicz R., Najgebauer A., Pierzchała D., Tarapata Z. (2009c): Systemy wspomagania dowodzenia w procesie planowania działań operacyjnych: problemy modelowania, projektowania i integracji, *Biuletyn Instytutu Systemów Informatycznych*, nr 3 (1/2009), ISBN 1508-4183, pp. 1-13.

Antkiewicz R., Najgebauer A., Tarapata Z., Rulka J. (2009d): Ocena rzeczywistej realizacji przyjętego wariantu działania, *Sprawozdanie z realizacji zadania badawczego nr 15203: Koncepcja opartego na zasobach wiedzy systemu wspomagania procesu podejmowania decyzji w operacji sieciocentrycznej. Podzadanie 7,* Program Badawczy Zamawiany

Nr PBZ–MNiSW–DBO–02/I/2007, Zaawansowane metody i techniki tworzenia świadomości sytuacyjnej w działaniach sieciocentrycznych, WIŁ-WAT-PIT-CTM--ABG, Warszawa.

Antkiewicz R., Chmielewski M., Kasprzyk R., Koszela J., Kręcikij J., Kulas W., Najgebauer A., Pierzchała D., Rulka J., Tarapata Z., Tarnawski T., Wantoch-Rekowski R. (2010a): Modelowanie przestrzeni walki, w: Amanowicz M. (red.) *Zaawansowane metody i techniki tworzenia świadomości sytuacyjnej w działaniach sieciocentrycznych*, Wydawnictwo PTM, Warszawa, pp. 360-382.

Antkiewicz R., Chmielewski M., Kasprzyk R., Koszela J., Kręcikij J., Kulas W., Najgebauer A., Pierzchała D., Rulka J., Tarapata Z., Tarnawski T., Wantoch-Rekowski R. (2010b): System wspomagania dowodzenia oparty na mechanizmach usług rozproszonych, w: Amanowicz M. (red.) *Zaawansowane metody i techniki tworzenia świadomości sytuacyjnej w działaniach sieciocentrycznych*, Wydawnictwo PTM, Warszawa, pp. 412-435.

Antkiewicz R., Chmielewski M., Kasprzyk R., Koszela J., Kręcikij J., Kulas W., Najgebauer A., Pierzchała D., Rulka J., Tarapata Z., Tarnawski T., Wantoch-Rekowski R. (2010c): Weryfikacja mechanizmów wspomagania procesu dowodzenia, w: Amanowicz M. (red.) *Zaawansowane metody i techniki tworzenia świadomości sytuacyjnej w działaniach sieciocentrycznych*, Wydawnictwo PTM, Warszawa, pp. 436-463.

Antkiewicz R., Chmielewski M., Kasprzyk R., Koszela J., Kręcikij J., Kulas W., Najgebauer A., Pierzchała D., Rulka J., Tarapata Z., Tarnawski T., Wantoch-Rekowski R. (2010d): Wspomaganie podejmowania decyzji w operacji sieciocentrycznej w oparciu o zasoby wiedzy, w: Amanowicz M. (red.) *Zaawansowane metody i techniki tworzenia świadomości sytuacyjnej w działaniach sieciocentrycznych*, Wydawnictwo PTM, Warszawa, pp. 383-411.

Antkiewicz R., Gąsecki A., Najgebauer A., Pierzchała D., Tarapata Z. (2010e): Stochastic PERT and CAST Logic Approach for Computer Support of Complex Operation Planning, *ASMTA'2010, Lecture Notes in Computer Science*, vol. 6148, Springer, Heidelberg, pp. 159-173.

Antkiewicz R., Kulas W., Najgebauer A., Pierzchała D., Rulka J., Tarapata Z., Wantoch-Rekowski R. (2010f): Selected Problems of Designing and Using Deterministic and Stochastic Simulators for Military Trainings, *43rd Hawaii International Conference on System Sciences,* IEEE Computer Society, ISBN 978-0--7695-3869-3, January 5-8, Koloa, Kauai, Hawaii (USA).

Antkiewicz R., Kulas W., Najgebauer A., Pierzchała D., Rulka J., Tarapata Z., Wantoch-Rekowski R. (2011a): Fast CoA Verification and Recommendation using Tactical Level Land Battle Deterministic Simulator, *Proceedings of the 13th International Conference on Computer Modelling and Simulation UkSim'2011*, IEEE Computer Society, Cambridge (UK), March 30–April 1, pp. 137-144.

Antkiewicz R., Najgebauer A., Rulka J., Tarapata Z., Wantoch-Rekowski R. (2011b): Knowledge-Based Pattern Recognition Method and Tool to Support Mission Planning and Simulation, *ICCCI'2011, Part I, Lecture Notes in Computer Science*, vol. 6922, Springer, Heidelberg, pp.478-487.

Atallah M., Chen D., Daescu O. (1997): Efficient Parallel Algorithms for Planar st-Graphs, *Lecture Notes in Computer Science*, vol. 1350, Springer, Heidelberg, pp. 223-232.

Barabási A.L., Réka A. (2002): Statistical Mechanics of Complex Networks, *Review of Modern Physics*, vol. 74, pp. 47-97.

Bartosiak C., Kasprzyk R., Tarapata Z. (2011): An Application of Graphs and Networks Similarity Methods to Networks Analyzing, *Biuletyn Instytutu Systemów Informatycznych*, No. 7 (in press).

Beautement, P., Allsopp, D., Greaves, M., Goldsmith, S., Spires, S., Thompson, S., Janicke, H. (2006): Autonomous Agents and Multi-agent Systems (AAMAS) for the Military - Issues and Challenges, *Lecture Notes in Computer Science*, vol. 3890, Springer, Heidelberg, pp. 1-13.

Behnke S. (2004): Local Multiresolution Path Planning, In B. Browning, D. Polani, A. Bonarini, and K. Yoshida (editors): *RoboCup: Robot Soccer World Cup VII, Lecture Notes in Computer Science*, vol. 3020, Springer, Heidelberg, pp. 332-343.

Bellman R. (1958): On a Routing Problem, *Quarterly of Applied Mathematics*, vol. 16, pp. 87-90.

Benton J.R., Iyengar S.S., Deng W., Brener N., Subrahmanian V.S. (1995): Tactical Route Planning: New Algorithms for Decomposing the Map, *Proceedings of the IEEE International Conference on Tools for AI*, November 6-8, Herndon, pp. 268-277.

Berman O., Kara B. Y., Verter V. (2007): Designing Emergency Response Networks for Hazardous Materials Transportation, *Computers & Operations Research,* vol. 34, pp. 1374–1388.

Bernstein D., Kelly S. (1997): Solving a Best Path Problem When the Value of Time Function is Nonlinear, *Annual Meeting of the Transportation Research Board*, preprint 980976, Princeton University.

Bhandari R. (1999): Survivable Networks. Algorithms for Divers Routing, *Kluver Academic Publishers*, Boston/Dordrecht/London.

Bianco L., Caramia M., Giordani S. (2009): A Bilevel Flow Model for Hazmat Transportation Network Design, *Transportation Research*, vol. C 17, pp. 175-196.

Blondel V., Gajardo A., Heymans M., Senellart P., Van Dooren P. (2004): A Measure Of Similarity Between Graph Vertices: Applications To Synonym Extraction And Web Searching, *SIAM Review*, vol. 46(4), pp. 647–666.

Brodal G., Jacob R. (2004): Time-Dependent Networks as Models to Achieve Fast Exact Time-Table Queries, *Electronic Notes in Theoretical Computer Science*, vol. 92, pp. 3-15.

Brumbaugh-Smith J., Shier D. (1989): An Empirical Investigation of Some Bicriterion Shortest Path Algorithms, *European Journal of Operational Research*, vol. 43(2), pp. 216-224.

Buchli J. (edt.) (2006): Mobile Robotics Moving Intelligence, *Pro Literatur Verlag*, ISBN 3-86611-284-X, Germany.

Bunke H. (1997): On a Relation Between Graph Edit Distance and Maximum Common Subgraph, *Pattern Recognition Letters*, vol. 18, pp. 689–694.

Bunke H. (2000): Graph Matching: Theoretical Foundations, Algorithms, and Applications, in *Proceedings Vision Interface* , Montreal, pp. 82-88.

Busacker R.G., Gowen P.J. (1961): A Procedure for Determining a Family of Minimum-Cost Flow Patterns, *Operations Research Office Technical Report 15*, John Hopkins University, Baltimore.

Cai X., Kloks T., Wong C.K. (1997): Time-Varying Shortest Path Problems with Constraints, *Networks*, vol. 29, pp. 141-149.

Campbell C., Hull R., Root E., Jackson L. (1995): Route Planning in CCTT, in *Proceedings of the 5th Conference on Computer Generated Forces and Behavioural Representation*, Technical Report, Institute for Simulation and Training, pp. 233-244.

Caramia M., Guerriero F. (2009): A Heuristic Approach to Long-Haul Freight Transportation with Multiple Objective Functions, *Omega*, vol. 37, pp. 600-614.

Carotenuto P., Giordani S., Ricciardelli S., Rismondo S. (2007a): A Tabu Search Approach for Scheduling Hazmat Shipments, *Computers & Operations Research*, vol. 34, pp. 1328–1350.

Carotenuto P., Giordani S., Ricciardelli S. (2007b): Finding Minimum and Equitable Risk Routes for Hazmat Shipments, *Computers & Operations Research*, vol. 34, pp. 1304–1327.

Carraway R.L., Morin T.L., Moskovitz H. (1990): Generalized Dynamic Programming for Multicriteria Optimization, *European Journal of Operational Research*, vol. 44, pp. 95-104.

Cassandras C.G., Panayiotou C.G., Diehl G., Gong W-B., Liu Z., Zou C. (2000): Clustering Methods for Multi-Resolution Simulation Modeling, *Proceedings of the Conference on Enabling Technology for Simulation Science*, The International Society for Optical Engineering, April 25-27, Orlando (USA), pp. 37-48.

CBS (2001): Corps Battle Simulation (CBS), Version 1.6.0, COBRA Users Guide, *U.S. Army Simulation, Training, and Instrumentation Command*, Orlando, Florida.

Ceranowicz A. (1994): Modular Semi-Automated Forces, *Proceedings of the Winter Simulation Conference*, Orlando, Florida.

Champin P., Solnon Ch. (2003): Measuring The Similarity of Labeled Graphs, *Proc. 5th International Conference on Case-Based Reasoning (ICCBR'2003), Lecture Notes in Artificial Intelligence*, vol. 2689, Springer, Heidelberg, pp. 80–95.

Chen Y.-W., Wang Ch.-H., Lin S.-J. (2008): A Multi-Objective Geographic Information System for Route Selection of Nuclear Waste Transport, *Omega*, vol. 36, pp. 363-372.

Cherkassky B. V., Goldberg A. V., Radzik T. (1996): Shortest Paths Algorithms: Theory and Experimental Evaluation, *Mathematical Programming,* vol. 73, pp. 129-174.

Cheung Ch., Fung S., Kwok S., Lee W., Tan B.: (2007): A Study of Knowledge-Based Simulation for Enterprise Resources Planning, *Journal of Information & Knowledge Management*, vol. 6(4), pp. 241-249.

Chmielewski M. (2008a): Data Fusion Based on Ontology Model for Common Operational Picture Using OpenMap and Jena Semantic Framework, *Proceedings of the Military Communications and Information Systems Conference MCC'2008*, ISBN 83-920120-5-4, September 23-24, Cracow, Poland.

Chmielewski M., Kasprzyk R. (2008b): Usage and Characteristics of Ontology Models in Network Enabled Capability Operations, *Proceedings of the Military Communications* *and Information Systems Conference MCC'2008*, ISBN 83-920120-5-4, September 23-24, Cracow, Poland.

Chmielewski M., Galka A. (2010a): Semantic Battlespace Data Mapping Using Tactical Symbology, in: *Advances in Intelligent Information and Database Systems. Studies in Computational Intelligence*, N.T. Nguyen et al. (eds.), Springer, Heidelberg, vol. 283, pp. 157-168.

Chmielewski M., Nogalski D. (2010b): Semantic Web Service Discovery and Information Fusion Using OWL-S and SPARQL Formal Specifications over NATO JC3IEDM and TIDE Services, in: *Concepts and Implementations for Innovative Military Communications and Information Technologies*, M. Amanowicz (edt.), WAT, Warsaw, pp. 165-174.

Chmielewski M., Wilkos M., Wilkos K. (2010c): Building Multiagent for Military Decision Support Tools with Semantic Services, *KES-AMSTA'2010, Lecture Notes in Artificial Intelligence*, vol. 6070, Springer, Heidelberg, pp. 173-182.

Chou Y., Romeijn H. E., Smith R. L. (1998): Approximating Shortest Paths in Large-scale Networks with an Application to Intelligent Transportation Systems, *INFORMS Journal on Computing*, vol. 10(2), pp. 163-179.

Cidon I., Rom R., Shavitt Y. (1997): Multi-Path Routing Combined with Resource Reservation, *Proceedings of the 16th Annual Joint Conference of the IEEE Computer and Communications Societies INFOCOM '97*, Kobe (Japan), pp. 92-100.

Cidon I., Rom R., Shavitt Y. (1999): Analysis of Multi-Path Routing, *IEEE/ACM Transactions on Networking*, vol. 7(6), pp. 885-896.

Cil, B., Mala, M. (2010): A Multi-Agent Architecture for Modelling and Simulation of Small Military Unit Combat in Asymmetric Warfare, *Expert Systems with Applications*, vol. 37(2), pp. 1331-1343.

Climaco J., Martins E. (1982): A Bicriterion Shortest Path Algorithm, *European Journal of Operational Research*, vol.11, pp. 399-404.

Clímaco, J., Craveirinha, J., Pascoal, M. (2002): A Bicriterion Approach for Routing Problems in Multimedia Networks, *Networks*, vol. 41(4), pp.206-220.

Cooke K., Halsey E. (1996): The Shortest Route Through a Network with Time-Dependent Intermodal Transit Times, *Journal Math. Anal. Appl.*, vol. 14, pp. 493-498.

Corea G.A., Kulkarni V. G. (1990): Minimum Cost Routing on Stochastic Networks, *Operations Research*, vol. 38, pp. 527-536.

Corley H.W., Moon I.D. (1985): Shortest Paths in Networks with Vector Weights, *Journal of Optimization Theory and Applications*, vol. 46(1), pp. 79-86.

Cormen T., Leiserson CH.E., Rivest R.L. (1994): Introduction to Algorithms, *Massachusetts Institute of Technology*, Cambridge.

Cormican K., Morton D., Wood K. (1998): Stochastic Network Interdiction, *Operations Research,* vol. 46, pp. 184-197.

Courtemanche A. J., Monday P. (1994): The Incorporation of Validated Combat Models into ModSAF, *Fourth Conference on Computer Generated Forces and Behavioral Representation*, University of Central Florida, Orlando.

Coutinho-Rodrigues J.M., Climaco J.C.N., Current J.R. (1999): An Intercative Biobjective Shortest Path Approach: Searching for Unsupported Nondominated Solutions, *Computers and Operations Research*, vol. 26(8), pp. 789-798.

Cox R.G. (1984): Routing of Hazardous Material, PhD thesis, *School of Civil and Environmental Engineering*, Cornell University, Ithaca , NY.

Crauser A., Mehlhorn K., Meyer U., Sanders P. (1998): A Parallelization of Dijkstra's Shortest Path Algorithm, *Lecture Notes in Computer Science*, vol. 1450 , pp. 722-731.

Current J.R., ReVelle C.S., Cohon J.L. (1990): An Interactive Approach to Identify the Best Compromise Solution for Two Objective Shortest Path Problems, *Computers and Operations Research*, vol. 17(2), pp. 187-198.

Davis P.K., Bigelow J.H., McEver J. (2000): Informing and Calibrating a Multiresolution Exploratory Analysis Model with High Resolution Simulation: the Interdiction Problem as a Case History, *Proceedings of the Winter Simulation Conference*, Orlando (FL, USA), pp. 316-325.

Dean B. (2004): Algorithms for Minimum-Cost Paths in Time-Dependent Networks with Waiting Policies, *Networks*, vol. 44(1), pp. 41–46.

Dial R. (1979): A Model and Algorithm for Multicriteria Route-Mode Choice, *Transportation Research*, vol. 13B, pp. 311-316.

Dijkstra E. (1959): A Note on Two Problems in Connection with Graphs, *Numerische Mathematik*, vol. 1, pp. 269-271.

Djidjev H., Pantziou G., Zaroliagis C. (1995): On-Line and Dynamic Algorithms for Shortest Path Problems, *Lecture Notes in Computer Science*, vol. 900, Springer, Heidelberg, pp. 193-204.

Dockery J., Woodcock A.E.R. (1993): The Military Landscape, Mathematical Models of Combat, *Woodhead Publishing Ltd.*, Cambridge, England.

Dompke U. (2001): Computer Generated Forces - Background, Definition and Basic Technologies, RTO-EN-017, *SAS Lecture Series on "Simulation of and for Military Decision Making"*, Rome, Italy, October 15-16.

Duckham M., Kulik L. (2003): "Simplest" Paths: Automated Route Selection for Navigation, in: W. Kuhn, M.F. Worboys, S. Timpf (Eds), Spatial Information Technology: Foundations of Geographic Information Science, *Lecture Notes in Computer Science*, vol. 2825, Springer, Heidelberg, pp. 182-199.

Duran E., Costaguta R. (2009): The Knowledge-Based Simulation, *Proceedings of the 31st Annual Simulation Symposium*, Boston, Massachusetts, April 5.

Edmonds J., Karp R.M. (1972): Theoretical Improvement in Algorithmic Efficiency for Network Flow Problems, *J. ACM*, vol. 19 , pp. 248-264.

Ehrgott M. (1997): Multiple Criteria Optimization - Classification and Methodology, *Shaker Verlag*, Aachen.

Ehrgott M., Gandibleux X. (Eds) (2002): Multiple Criteria Optimization – State of the Art Annotated Bibliographic Surveys, *Kluwer Academic Publishers*, Boston.

Engberg D., Cohon J., ReVelle C. (1983): Multiobjective Siting of a Natural Gas Pipeline, Instrument Society of America, *Proceedings of the 11th Annual Pittsburgh Conference on Modeling and Simulation,* Pittsburgh.

Eppstein D. (1995): Finding Common Ancestors and Disjoint Paths in DAGs, *Technical Report 95-52*, Department of Information and Computer Science, University of California, Irvine.

Eppstein D. (1999): Finding the K Shortest Paths, *SIAM Journal of Computing*, vol. 28(2), pp. 652-673.

Ergun F., Sinha R., Zhang L. (2002): An Improved FPTAS for Restricted Shortest Path, *Information Processing Letters*, vol. 83, pp. 287-291.

Erkut E., Kara B.Y., Verter V. (2003): Accurate Calculation of Hazardous Materials Transport Risks, *Operations Research Letters*, vol. 31, pp. 285-292.

Eschenauer H., Koski J., Osyczka A. (1990): Multicriteria Design Optimization, *Springer Verlag*, Berlin-Heidelberg-New York.

Even S., Itai A., Shamir A. (1976): On the Complexity of Time-Table and Multicommodity Flow Problems, *SIAM Journal on Computing*, vol. 5, pp. 691-703.

Farin D., With P., Effelsberg W. (2003): Recognition of User-Defined Video Object Models using Weighted Graph Homomorphisms, *Proc. Image and Video Communications and Processing (IVCP'2003), Proc. SPIE 5022,* Bhaskaran Vasudev, T. Russell Hsing, Andrew G. Tescher, and Touradj Ebrahimi, ed., pp. 542–553.

Foster I. (1995): Designing and Building Parallel Programs, *Addison-Wesley*.

Frank H. (1969): Shortest Paths in Probabilistic Graphs, *Operations Research*, vol. 17, pp. 583-599.

Fredman M.L., Tarjan R.E. (1987): Fibonacci Heaps and their Uses in Improved Network Optimization Algorithms, *Journal of the Association for Computing Machinery*, vol. 34, pp.596–615.

Fujimura, K. (1996): Path Planning with Multiple Objectives, *IEEE Robotics and Automation Magazine*, vol. 3, pp. 33–38.

Gabow H.N., Tarjan R.E. (1989): Faster Scaling Algorithms for Network Problems, *SIAM Journal on Computing*, No. 18, pp. 1013-1036.

Gabrel V., Vanderpooten D. (1996): Generation and Selection of Efficient Paths in a Multiple Criteria Graph: The Case of Daily Planning The Shots Taken by a Satellite with an Interactive Procedure, *Technical Report 136,* LAMSADE, Universitae Paris Dauphine.

Garey M., Johnson D. (1979): Computers and Intractability: A Guide to the Theory of NP Completeness, *W.H.Freeman*, San Francisco.

Gelenbe, E., Kaptan, V., Hussain, K. (2004): Simulating the Navigation and Control of Autonomous Agents, *Proceedings of the 7th International Conference on Information Fusion*, Stockholm, pp. 183-189.

Gilmore J., Semeco A. (1985): Terrain Navigation Through Knowledge-Based Route Planning, *Proceeding of the 9th international joint conference on Artificial intelligence (IJCAI'85)*, vol. 2, pp. 1086-1088.

Godlewski P. (2010): Modelowanie i algorytmizacja procesów planowania i symulacji przemieszczania z wykorzystaniem wielorozdzielczych modeli terenu, *praca magisterska pod kierunkiem Z.Tarapaty*, Wojskowa Akademia Techniczna, Warszawa.

Golden B.L., Skiscim C.C. (1989): Solving K-Shortest and Constrained Shortest Path Problems Efficiently, *Network Optimization and Applications*, vol. 20, Texas A&M University, College Station.

Grama A., Gupta A., Karypis G., Kumar V. (2003): Introduction to Parallel Computing, *Addison-Wesley*, ISBN 0-2016-4865-2.

Ground L., Kott A., Budd R. (2002): A Knowledge-Based Tool for Planning of Military Operations: the Coalition Perspective, *BBN Technologies*, Pittsburgh.

Grzech A. (2002): Sterowanie ruchem w sieciach teleinformatycznych, *Oficyna Wydawnicza Politechniki Wrocławskiej*, Wrocław.

Guo Y., Yang M., Cheng J. (2010): Knowledge-Inducing Global Path Planning for Robots in Environment with Hybrid Terrain, *International Journal of Advanced Robotic Systems*, vol. 7(3), pp. 239-248.

Guru (2005): Opracowanie projektu systemu eksperckiego z modelowym oprogramowaniem bazowym ZNWD, *Projekt wstępny "Zautomatyzowane narzędzia wspomagania decyzji – system ekspercki pk. Guru"*, t. I. 6, Wydział Cybernetyki, WAT, Warszawa.

Halder D.K., Majumber A. (1981): A Method for Selecting Optimum Number of Stations For a Rapid Transit Line: An Application in Calcutta Tube Rail, in N.K. Jaiswal, editor, *Scientific Management of Transport Systems*, pp. 97-108.

Han Y., Pan V., Reif J. (1997): Efficient Parallel Algorithms for Computing All Pair Shortest Paths in Directed Graphs, *Algorithmica*, vol. 17 , pp. 399-415.

Hansen P. (1979): Bicriterion Path Problems, in: G. Fandel and T. Gal (eds.), Multiple Criteria Decision Making Theory and Application, *Lecture Notes in Economics and Mathematical Systems*, vol. 177, pp. 109-127, Springer, Heidelberg.

Hart P.E., Nilsson N.J., Raphael B. (1968): A Formal Basis for the Heuristic Determination of Minimum Cost Paths, *IEEE Trans. Syst. Scien. Cybern.*, vol. 4(2), pp. 100-107.

Hartley R. (1985): Vector Optimal Routing by Dynamic Programming, in P. Serahni, editor, *Mathematics of Multiobjective Optimization*, vol. 289 of CISM International Centre for Mechanical Sciences, Courses and Lectures, Springer Verlag, Wien, pp. 215-224.

Hassin R. (1992): Approximation Schemes for the Restricted Shortest Path Problem, *Mathematics of Operations Research*, vol. 17, pp. 36-42.

Heal, G. N., Garnett I.K. (2001): Allied Deployment and Movement System (ADAMS) Version 3.0 Tutorial, *Technical note 669, NATO consultation, Command and Control Agency*, Hague (The Netherlands).

Heero K. (2006): Path Planning And Learning Trategies For Mobile Robots In Dynamic Partially Unknown Environments, *PhD Thesis*, Faculty of Mathematics and Computer Science, University of Tartu, Estonia.

Henig M.I. (1985): The Shortest Path Problem with Two Objective Functions, *European Journal of Operational Research*, vol. 25, pp. 281-291.

Henig M.I. (1994): Efficient Interactive Methods for a Class of Multiattribute Shortest Path Problems, *Management Science*, vol. 40(7), pp. 891-897.

Henninger A.E., Gonzalez A.J., Georgipoulos M., DeMara R.F. (2000): Modeling Semi-Automated Forces with Neural Networks: Performance Improvement through a Modular Approach, *The Ninth Conference on Computer Generated Forces and Behavioral Representation Proceedings*, Orlando (FL, USA).

Hodal J., Dvorak J. (2008): Using Case-Based Reasoning for Mobile Robot Path Planning, *Engineering Mechanics*, vol. 15(3), pp. 181-191.

Hofmann M. (2005): On the Complexity of Parameter Calibration in Simulation Models, *JDMS*, The Society for Modeling and Simulation International, vol. 2(4), pp. 217-226.

Hofmann H.W., Hofmann M. (2000): On the Development of Command, Control Modules for Combat Simulation Models on Battalion down to Single Item Level, in: *Proceedings of the NATO/RTO Information Systems Technology Panel (IST) Symposium "New Information Processing Techniques for Military Systems"*, NATO AC/329 (IST--017) TP/8, Istanbul, Turkey, October 9-11, pp. 85-96.

Holsapple C., Whinston A. (1996): Decision Support Systems: a Knowledge-Based Approach, *St. Paul: West Publishing*, ISBN 0-324-03578-0.

Hu Y., Yang S., Xu L., Meng M. (2004): A Knowledge Based Genetic Algorithm for Path Planning in Unstructured Mobile Robot Environments, *Proceedings of the 2004 IEEE International Conference on Robotics and Biomimetics*, August 22-26, Shenyang, China, pp. 767-772.

Huarng F., Pulat P.S., Shih L. (1996): A Computational Comparison of Some Bicriterion Shortest Path Algorithms, *Journal of the Chinese Institute of Industrial Engineers*, vol. 13(2), pp. 121-125.

Ibaraki T. (1973): Algorithms for Obtaining Shortest Paths Visiting Specified Nodes, *SIAM Review*, vol. 15, No. 2, Part 1, pp. 309-317.

Ibaraki T., Katon N., Mine H. (1978): 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.*, vol. 12, pp. 1199-1206.

Jacyna M. (2009): Wybrane zagadnienia modelowania systemów transportowych, *Oficyna Wydawnicza Politechniki Warszawskiej*, Warszawa.

James J., Sayrs B., Benton J., Subrahmanian V.S. (1999): Uncertainty Management: Keeping Battlespace Visualization Honest, *Proceedings of the 3rd Annual Conference on Advanced Telecommunications and Information Distribution Research Program (ATIRP)*, February 1-5, University of Maryland, College Park, MD.

Jing X-J. (2008): Motion Planning, *InTech Education and Publishing*, ISBN 978-953-7619-01-5, Vienna (Austria).

Joe L., Feldman P.M. (1998): Fundamental Research Policy for the Digital Battlefield, *Research Report DB-245-A*, RAND Co., Santa Monica (USA).

Johnson D.B. (1977): Efficient Algorithm for Shortest Paths in Sparse Networks, *Journal of the ACM*, vol. 24, pp. 1-13.

Jongh A., Gendreau M., Labbe M. (1999): Finding Disjoint Routes in Telecommunications Networks with Two Technologies, *Operations Research*, vol. 47, pp. 81-92.

JTLS (1988): The Analyst Guide, Joint Theater Level Simulation (JTLS), Version 1.65, Modern Aids to Planning Program (MAPP), *Force Structure and Assessment Directorate (J-8)*, Joint Staff, The Pentagon.

Kambhampati S., Davis L.S. (1986): Multiresolution Path Planning for Mobile Robots, *IEEE Journal of Robotics and Automation*, vol. RA-2, No. 3 , pp. 135-145.

Karr C.R., Craft M.A., Cisneros J.E. (1995): Dynamic Obstacle Avoidance, *Proceedings of the Conference on Distributed Interactive Simulation Systems for Simulation and Training in the Aerospace Environment*, The International Society for Optical Engineering, April 19-20, Orlando (USA), pp. 195-219.

Kasprzyk R. (2005): Complex Networks in Countering Terrorism*, Proceedings of the International PHD Workshop OWD'2005*, Conference Archives PTETiS, vol. 21, ISBN 83-922242-0-5, pp.95-100.

Kasprzyk R. (2008): Fault and Attack Resistance of Complex Networks, *Proceedings of the International PHD Workshop OWD'2008*, Conference Archives PTETiS, vol. 25, ISBN 83-922242-0-5, pp.205-210.

Kaufman D. E., Smith R. L. (1993): Fastest Paths in Time-Dependent Networks for Intelligent Vehicle Highway Systems Applications, *IVHS Journal*, vol.1(1), pp.1-11.

Kerbache L., Smith J. (2000): Multi-Objective Routing Within Large Scale Facilities Using Open Finite Queueing Networks, *European Journal of Operational Research*, vol. 121, pp. 105-123.

Kleinberg J. (1999): Authoritative Sources in Hyperlinked Environment, *Journal of the ACM*, vol. 46(5), pp. 604–632.

Klupfel H., Schreckenberg M., Meyer-Konig T. (2005): Models for Crowd Movement and Egress Simulation, in: *Traffic and Granular Flow '03*, Springer, Heidelberg, pp. 357-372.

Korf R.E. (1999): Artificial Intelligence Search Algorithms, in *Algorithms Theory Computation Handbook*, Boca Raton, FL: CRC Press.

Kornell J. (1987): Reflections on Using Knowledge Based Systems for Military Simulation, *Simulation*, vol. 48, pp. 144-148.

Korzan B. (1978): Elementy teorii grafów i sieci, Metody i zastosowania, *WNT*, Warszawa.

Korzan B. (1982): Metoda wyznaczania dróg kompromisowych w zawodnych sieciach skierowanych, *Biuletyn Wojskowej Akademii Technicznej*, vol. 7, pp. 21-36.

Korzan B. (1983a): Metoda wyznaczania dróg niezdominowanych w zawodnych sieciach skierowanych, *Biuletyn Wojskowej Akademii Technicznej,* vol. 11, pp. 21-33.

Korzan B. (1983b): Optymalizacja dróg w zawodnych sieciach skierowanych, *Biuletyn Wojskowej Akademii Technicznej*, vol. 6, pp. 69-85.

Kostreva M.M. , Wiecek M.M. (1993): Time Dependency in Multiple Objective Dynamic Programming, *Journal of Mathematical Analysis and Applications*, vol. 173(1), pp. 289-307.

Koszela J., Chmielewski M. (2008): The Concept of C4I Systems Data Integration for Planning Joint Military Operations Based on JC3 Standard, *Proceedings of the Military Communications and Information Systems Conference MCC'2008*, ISBN 83--920120-5-4, September 23-24, Cracow, Poland.

Krebs V. (2002): Mapping Networks of Terrorist Cells, *Connections*, vol. 24(3), pp. 43-52.

Kreitzberg T., Barragy T., Nevin B. (1990): Tactical Movement Analyzer: a Battlefield Mobility Tool, *Proceedings of the 4th Join Tactical Fusion Symposium*, Laurel, pp. 363-383.

Kriegel H.P., Schonauer S. (2003): Similarity Search in Structured Data, *Proc. 5th Int. Conf. on Data Warehousing and Knowledge Discovery (DaWaK'03)*, Prague, Czech Republic, in: *Lecture Notes in Computer Science*, vol.2737, Springer, Heidelberg, pp.309–319.

Kuhl F., Weatherly D., Dahman J. (1999): Creating Computer Simulation Systems – An Introduction to the High Level Architecture, *PH PTR*, ISBN 0-13-022511-8.

Kulas W., Pierzchała D., Tarapata Z., Tarnawski T. (2008): Finding Optimal Flight Path in a Segmented Airspace, *Proceedings of the Military Communications and Information Systems Conference MCC'2008*, ISBN 83-920120-5-4, September 23-24, Cracow, Poland.

Kumar V., Grama A., Gupta A., Karypis G. (1994): Introduction to Parallel Programming – Design and Analysis of Algorithms, *Banjamin Cummings Publishing*.

Lanthier M., Nussbaum D., Sack J. (2003): Parallel Implementation of Geometric Shortest Path Algorithms, *Parallel Computing*, vol. 29(10), pp. 1445-1479.

LaValle S. (2006): Planning Algorithms, *Cambridge University Press*, Cambridge (UK).

Lee.J.J., Fishwick P.A. (1995): Simulation-Based Real-Time Decision Making for Route Planning, *Proceedings of Winter Simulation Conference*, Orlando (FL, USA), pp. 1087-1095.

Lee J.J. (1996): A Simulation-Based Approach for Decision Making and Route Planning, *PhD Thesis*, University of Florida, August.

Leung J.Y-T. (edt.) (2004): Handbook of Scheduling: Algorithms, Models and Performance Analysis, *Chapman, Hall/CRC*, Boca Raton-London-New York-Washington.

Li C.L., McCormick S.T., Simchi-Levi D. (1990): The Complexity of Finding Two Disjoint Paths with Min-Max Objective Function, *Discrete Applied Math*, vol. 26, pp. 105-115.

Li C.L., McCormick S.T., Simchi-Levi D. (1992): Finding Disjoint Paths with Different Path-Costs: Complexity and Algorithms, *Networks*, vol. 22, pp. 653-667.

Logan B. (1997a): Route Planning with Ordered Constraints, *Proceedings of the 16th Workshop of the UK Planning and Scheduling Special Interest Group*, December, Durham (UK), pp. 133-144.

Logan B., Sloman A. (1997b): Agent Route Planning in Complex Terrains, *Technical Report CSRP-97-30*, University of Birmingham, School of Computer Science, Birmingham.

Longtin M., Megherbi D. (1995): Concealed Routes in ModSAF, in *Proceedings of the 5th Conference on Computer Generated Forces and Behavioural Representation*, Orlando (FL, USA), pp. 305-314.

Lorenz D.H., Raz D. (2001): A Simple Efficient Approximation Scheme for the Restricted Shortest Path Problem, *Operations Research Letters*, vol. 28, pp. 213-219.

Loui R.P. (1983): Optimal Paths in Graphs with Stochastic or Multidimensional Weights, *Comm. Assoc. Comp. Mach.*, vol. 26, pp. 670-676.

Madni A.M., Ahlers R., Chu Y. (1987): Knowledge-Based Simulation: An Approach to Intelligent Opponent Modeling for Training Tactical Decisionmaking, *Proceedings of the 9th Interservice/Industry Training Systems Conference*, pp. 179-183, November 30-December 2, Washington, D.C.

Magillo P., Bertocci V. (1998): Managing Large Terrain Data Sets with a Multiresolution Structure, *INFORMS Journal on Computing*, vol. 10(2), pp. 163-179.

Mark D. (1986): Automated Route Selection for Navigation, *IEEE Aerospace and Electronic Systems Magazine*, vol. 1(9), pp. 2-5.

Martins E.Q.V., Climaco J.C.N. (1981): On the Determination of the Nondominated Paths in a Multiobjective Network Problem, *Methods of Operations Research*, vol. 40, pp. 255–258.

Martins E.Q.V. (1984): On a Multicriteria Shortest Path Problem, *European Journal of Operational Research*, vol.16, pp. 236-245.

Martins E.Q.V., Santos J.L.E. (1999): The Labelling Algorithm for the Multiobjective Shortest Path Problem, *Internal Technical Report, TR 1999/005, CISUC*, Universidade de Coimbra, Coimbra (Portugal).

Melnik S., Garcia-Molina H., Rahm E. (2002): Similarity Flooding: A Versatile Graph Matching Algorithm and its Application to Schema Matching, *Proceedings of the 18th International Conference on Data Engineering*, San Jose, California, pp. 117–128.

Meyer U., Sanders P. (2001): Parallel Shortest Paths for Arbitrary Graphs, *Lecture Notes in Computer Science*, vol. 1900, Springer, Heidelberg, pp. 461-470.

Mitchell J.S.B. (1999): Geometric Shortest Paths and Network Optimization, in J.R. Sack, J. Urrutia: *Handbook of Computational Geometry*, Elsevier Science Publishers, B.V. North-Holland, Amsterdam.

Modesti P., Sciomachen A. (1998): A Utility Measure for Finding Multiobjective Shortest Paths in Urban Multimodal Transportation Networks, *European Journal of Operational Research*, vol. 111(3), pp. 495-508.

Modsim (1994): MODSIM II. The Language for Object-Oriented Programming, Reference Manual, *CACI Products Company*, 1994.

Moffat J. (2003): Complexity Theory and Network Centric Warfare, *CCRP Publication Series*, ISBN 1-893723-11-9, Washington.

Mohn H. (1994): Implementation of a Tactical Mission Planner for Command and Control of Computer Generated Forces in ModSaf, *M. S. Thesis*, Naval Postgraduate School, Monterey.

Montana, D., Herrero, J., Vidaver, G., Bidwell, G. (2000): A Multiagent Society for Military Transportation Scheduling, *Journal of Scheduling*, vol. 3(4), pp. 225-246.

Mote J., Murthy I., Olson D. (1991): A Parametric Approach to Solving Bicriterion Shortest Path Problems, *European Journal of Operational Research*, vol. 53, pp. 81-92.

Murthy I., Her S.S. (1992): Solving Min-Max Shortest-Path Problems on a Network, *Naval Research Logistics*, vol. 39, pp. 669-683.

Murthy, Olson D. (1994): An Interactive Procedure Using Domination Cones for Bicriterion Shortest Path Problems, *European Journal of Operational Research*, vol. 72(2), pp. 418-432.

Nagarajan V., Raja P. (2010): Path Planning for Space Robots: Based on Knowledge Extrapolation and Risk Factors, *Proceedings of the 2010 IEEE International Conference on Automation and Logistics*, August 16-20, Hong Kong and Macau, pp. 373-378.

Najgebauer A. (1999a): Informatyczne systemy wspomagania decyzji w sytuacjach konfliktowych. Modele, metody i środowiska symulacji interaktywnej, *Suplement do Biuletynu Wojskowej Akademii Technicznej*, ISBN 83-908620-6-9, Warszawa.

Najgebauer A., Pierzchała D., Rulka J. (1999b): The Simulation Researches of Decision Processes in a Conflict Situation with Opposite Objectives, *Conference Proceedings of 13th European Simulation Multiconference ESM99*, Warsaw, Poland, June 1-4.

Najgebauer A. (2004a): Polish Initiatives in M&S and Training. Simulation Based Operational Training Support System (SBOTSS) Zlocien, *Proceedings of the ITEC'2004*, London, UK, April 20-22.

Najgebauer A. (2004b): Założenia i realizacja Symulacyjnego Systemu Wspomagania Szkolenia Operacyjnego dla Wojsk Lądowych SZ RP, *Materiały XII Konferencji Naukowej „Automatyzacja Dowodzenia"*, 2-4 czerwca, Gdynia-Jurata (publikacja na CD).

Najgebauer A., Antkiewicz R., Kulas W., Pierzchała D., Rulka J., Tarapata Z., Chmielewski M. (2004c): A Concept of Simulation Based Diagnostic Support Tool for Terrorism Threat Awareness, *Proceedings of NATO Modelling and Simulation Group Conference*, Koblenz, Germany, October 7-8.

Najgebauer A., Tarapata Z. (2004d): A Terrain Classification Method for Decision Automata in Simulation Aided System for Operational Training, *Proceedings of The 6th NATO Regional Conference on Military Communication and Information Systems*, ISBN 83-920120-0-3, October 6-8, Zegrze (Poland), pp. 111-116.

Najgebauer A., Antkiewicz R., Tarapata Z., Rulka J., Pierzchala D., Kulas W., Wantoch-Rekowski R. (2005): Case-Based C2 Modelling and Effective Development, Implementation and Experimentation for Simulation Based Operational Training Support System, in the Effectiveness of Modelling and Simulation – From Anecdotal to Substantive Evidence (pp. 6-1, 6-16), *Meeting Proceedings RTO-MP-MSG-035*, ISBN 92-837-0047-3, October 13-14, Warsaw, Paper 6. Neuilly-sur-Seine, France: RTO.

Najgebauer A., Antkiewicz R., Chmielewski M., Kasprzyk R. (2007a): The Prediction of Terrorist Threat on the Basis of Semantic Associations Acquisition And Complex Network Evolution. *Proceedings of the Military Communications and Information Systems Conference MCC'2007,* Bonn, Germany.

Najgebauer A., Antkiewicz R., Tarapata Z., Rulka J., Kulas W., Pierzchala D., Wantoch-Rekowski R. (2007b): The Automation of Combat Decision Processes in the Simulation Based Operational Training Support System, *Proceedings of the IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA'07)*, April 1-5, ISBN 1-4244-0698-6, Honolulu (Hawaii).

Najgebauer A. (2008a): Decision Support Systems in the Area of Security and Defense Using the Simulation and Artificial Intelligence Techniques, *Proceedings of the Military Communications and Information Systems Conference MCC'2008,* ISBN 83-920120-5-4, September 23-24, Cracow, Poland.

Najgebauer A., Antkiewicz R., Kulas W., Pierzchała D., Rulka J., Tarapata Z., Wantoch-Rekowski R. (2008b): Modelowanie i symulacja procesów dowodzenia w systemie symulacyjnego wspomagania szkolenia operacyjnego, w: Kasprzyk J., Najgebauer A., Sienkiewicz P. (red), *Badania operacyjne i systemowe a zagadnienia społeczeństwa informacyjnego, bezpieczeństwa i walki*, ISBN 83-894-7518-9, PAN IBS, Warszawa , pp. 339-246.

Najgebauer A., Antkiewicz R., Kulas W., Pierzchała D., Rulka J., Tarapata Z., Wantoch--Rekowski R., Koszela J., Tarnawski T. (2008c): Symulacyjny model działań bojowych szczebla operacyjnego i taktycznego, w: Kasprzyk J., Najgebauer A., Sienkiewicz P. (red), *Badania operacyjne i systemowe a zagadnienia społeczeństwa informacyjnego, bezpieczeństwa i walki*, ISBN 83-894-7518-9, PAN IBS, Warszawa, pp. 253-266.

Najgebauer A., Tarapata Z., Chmielewski M., Kasprzyk R. (2008d): Integracja systemów dowodzenia SZ RP, w: Mierczyk Z. (red.): *Nowoczesne technologie systemów uzbrojenia*, ISBN 978-83-89399-93-9, Wojskowa Akademia Techniczna, Warszawa, pp. 88-101.

Najgebauer A., Antkiewicz R., Rulka J., Tarapata Z., Kapałka M. (2009): Zagrożenia dla porządku i bezpieczeństwa publicznego, w: A. Najgebauer (red.) *Modele zagrożeń aglomeracji miejskiej wraz z systemem zarządzania kryzysowego na przykładzie m.st. Warszawy*, ISBN 978-83-61486-22-0, WAT, Warszawa , pp. 563-584.

Newman Mark E. J. (2003): The Structure and Function of Complex Networks, *SIAM Review*, vol. 45(2), pp. 167-256.

OneSAF (2008): One Semi-Automated Forces (OneSAF), Operational Requirements Document, Version 1.1 (2008), *http://www.onesaf.net*.

Orda A., Rom R. (1990): Shortest-Path and Minimum Delay Algorithms in Networks with Time-Dependent Edge-Length, *Journal of the ACM*, vol. 37(3), pp. 607-625.

Orda A., Rom R. (1991): Minimum Weight Paths in Time-Dependent Networks, *Networks*, vol. 3, pp. 295-319.

Orda A., Rom R. (1996): Distributed Shortest-Path Protocols for Time-Dependent Networks, *Distributed Computing*, vol. 10(1), pp. 49-62.

Oren, T.I. (2001): Impact of Data on Simulation: From Early Practices to Federated and Agent-Directed Simulations, in: A. Heemink et al. (eds.) *Proc. of EUROSIM 2001*, June 26-29, Delft, the Netherlands.

Ozaki K., Asama H, Ishida Y., Matsumoto A., Yokota K., Kaetsu H., Endo I. (1993): Synchronized Motion by Multiple Mobile Robots Using Communication, in *Proceedings of the 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1164-1169.

Pai D.K., Reissell L.M. (1994): Multiresolution Rough Terrain Motion Planning, Department Of Computer Sciences, University of British Columbia, *Technical Report TR 94-33*, Vancouver.

Paige R., Kruskal C.P. (1985): Parallel Algorithms for Shortest Path Problems, in: *Proceedings of the International Conference on Parallel Processing*, pp. 14-19.

Pallottino S., Scutella M. (1998): Shortest Path Algorithms in Transportation Models: Classical and Innovative Aspects, in: (P. Marcotte and S. Nguyen, eds.) *Equilibrium and Advanced Transportation Modelling*, Kluwer, pp. 245-281.

Pantziou G., Spirakis P., Zaroliagis Ch. (1990): Efficient Parallel Algorithms for Shortest Paths in Planar Graphs, *Lecture Notes in Computer Science*, vol. 447, Springer, Heidelberg, pp. 288-300.

Papadimitriou C., Yannakakis M. (2000): On the Approximability of Trade-offs and Optimal Access of Web Sources, in *Proc. 41st Symp. on Foundations of Computer Science-FOCS*, IEEE Computer Society, Washington (USA), pp. 86-92.

Pelegrin B., Fernandez P. (1998): On the Sum-Max Bicriterion Path Problem, *Computers and Operations Research*, vol. 25(12), pp. 1043-1054.

Perl Y., Shiloach Y. (1978): Finding Two Disjoint Paths Between Two Pairs of Vertices in a Graph, *Journal of the ACM*, vol. 25, pp. 1-9.

Petty M.D. (1995): Computer Generated Forces in Distributed Interactive Simulation, *Proceedings of the Conference on Distributed Interactive Simulation Systems for Simulation and Training in the Aerospace Environment*, The International Society for Optical Engineering, April 19-20, Orlando (USA), pp. 251-280.

Pierzchała D. (2005): Biblioteka programowa wspomagająca wytwarzanie symulatorów dyskretno-zdarzeniowych w języku wysokiego poziomu, *Materiały konferencyjne XI Warsztatów Naukowych PTSK nt. Symulacja w badaniach i rozwoju*, Warszawa, ISBN/ISSN: 83 88229-80-X.

Pohl J., Chapman A., Pohl K., Primrose J., Wozniak A. (2003): Decision-Support Systems: Notions, Prototypes, and In-Use Applications With Emphasis on Military Applications, *Design Institute Report: CADRU-11-97*, Collaborative Agent Design Research Center (CADRC), California Polytechnic State University, San Luis Obispo (CA, USA).

Przemieniecki J.S. (1994): Mathematical Methods in Defence Analysis, *American Institute of Aeronautics and Astronautics*, Inc., Washington (DC, USA).

Rajput S., Karr C. (1994): Unit Route Planning, *Technical Report IST-TR-94-42*, Institute for Simulation and Training, Orlando (USA).

Rana K., Vickson R.G. (1988): A Model and Solution Algorithm for Optimal Routing of a Time-Chartered Containership, *Transportation Science*, vol. 22, pp. 83-96.

Reece D., Kraus M., Dumanoir P. (2000): Tactical Movement Planning for Individual Combatants, *Proceedings of the Conference on Computer Generated Forces and Behavioral Representation*, May 16-18, Orlando (USA).

Reece D. (2003): Movement Behavior for Soldier Agents on a Virtual Battlefield, Presence: Teleoperators and Virtual Environments, *MIT Press Journals*, vol. 12 (4), pp. 387-410.

Robinson S. (2004): The Ongoing Search for Efficient Web Search Algorithms, *SIAM News*, vol. 37(9), pp. 1-3.

Robinson S., Alifantis T., Edwards J.S., Ladbrook J., Waller T. (2005): Knowledge Based Improvement: Simulation and Artificial Intelligence for Identifying and Improving Human Decision-Making in an Operations System, *Journal of the Operational Research Society*, vol. 56(8), pp. 912-921.

Rosenthal R. (2010): GAMS - A User's Guide, *GAMS Development Corporation*, Washington, DC, USA.

Ross K., Klein G., Thunholm P., Schmitt J., Baxter H. (2004): The Recognition-Primed Decision Model, *Military Review*, July-August, pp. 6-10.

Rothenberg J., Narain S., Steeb R., Hefley Ch., Shapiro N. (1989): Knowledge-Based Simulation: an Interim Report, *RAND Note Report N-2897-DARPA*, RAND Corporation, Santa Monica (USA).

Sahin, C., Urrea, E., Uyar, M., Conner, M., Hokelek, I., Bertoli, G., Pizzo, Ch. (2008): Self-Deployment of Mobile Agents in Manets for Military Applications, In: *Proceedings of the 26th Army Science Conference,* Orlando (FL, USA), pp. 1-8.

Sancho N.G.F. (1988): A New Type of Multiobjective Routing Problem, *Engineering Optimization*, vol. 14, pp. 115-119.

Sawyer R. (1995): Sun Pin: Military Methods, *Westview Press*, Boulder, Colorado.

Schiavone G.A., Nelson R.S., Hardis K.C. (1995): Interoperability Issues for Terrain Databases in Distributed Interactive Simulation, *Proceedings of the Conference on Distributed Interactive Simulation Systems for Simulation and Training in the Aerospace Environment*, The International Society for Optical Engineering, April 19-20, Orlando (USA), pp. 89-120.

Schiavone G.A., Nelson R.S., Hardis K.C. (2000): Two Surface Simplification Algorithms for Polygonal Terrain with Integrated Road Features, *Proceedings of the Conference on Enabling Technology for Simulation Science*, The International Society for Optical Engineering, April 25-27, Orlando (USA), pp. 221-229.

Schrijver A., Seymour P. (1992): Disjoint Paths in a Planar Graph – A General Theorem, *SIAM Journal of Discrete Mathematics*, vol. 5, pp. 112-116.

Schrijver A. (2004): Combinatorial Optimization. Polyhedra and Efficiency, *Springer-Verlag*, Berlin- New York.

Senellart P., Blondel V. (2003): Automatic Discovery Of Similar Words, in: *Survey of Text Mining*, Springer-Verlag Berlin Heidelberg New York.

Sherali H., Ozbay K., Subramanian S. (1998): The Time-Dependent Shortest Pair of Disjoint Paths Problem: Complexity, Models and Algorithms, *Networks*, vol. 31, pp. 259-272.

Sigal E., Pritsker A., Solberg J. (1980): The Stochastic Shortest Route Problem, *Operations Research*, vol. 28, pp. 1122-1129.

Silva R., Craveirinha J. (2004): An Overview of Routing Models for MPLS Networks, *Proceedings of the First Workshop on Multicriteria Modelling in Telecommunication Network Planning and Design*, September, Faculty of Economics of the University of Coimbra.

Simgraphics (1995): Simgraphics II. User's Manual for MODSIM II, *CACI Products Company*, 1995.

Skriver A.J.V., Andersen K.A. (2000): A Label Correcting Approach for Solving Bicriterion Shortest Path Problems, *Computers and Operations Research*, vol. 27, pp. 507-524.

Smart P., Shadbolt N., Carr L., Schraefel M. (2005): Knowledge-Based Information Fusion for Improved Situational Awareness, *Proceedings of the 8th International Conference on Information Fusion*, July 25-29, Philadelphia, USA.

Sokolowski, J.A. (2002): Can a Composite Agent be Used to Implement a Recognition-Primed Decision Model?, in: *Proceedings of the Eleventh Conference on Computer Generated Forces and Behavioral Representation*, Orlando, FL., May 7-9, pp. 473-478.

Stentz A. (1994): Optimal and Efficient Path Planning for Partially-Known Environments, *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA'94*, vol. 4 , pp. 3310-3317.

Strogatz S.H. (2001): Exploring Complex Networks, *Nature*, vol. 410, pp. 268-276.

Subramanian S. (1995): Parallel and Dynamic Shortest-Paths Algorithms in Sparse Graphs, *PhD Thesis*, Department of Computer Science, Brown University, Providence.

Sun, T.Y., Tsai, S.J., Huo, Ch.L. (2008): Intelligent Maneuvering Decision System for Computer Generated Forces Using Predictive Fuzzy Inference System, *Journal Of Computers*, vol.3(11), pp. 58-66.

Suurballe J. W. (1974): Disjoint Paths in a Network, *Networks*, vol. 4, pp. 125-145.

Suurballe J.W., Tarjan R.E. (1984): A Quick Method for Finding Shortest Pairs of Disjoint Paths, *Networks*, vol. 14, pp. 325-336.

Tarapata Z. (1997): Algorithm for Simultaneous Finding a Few Independent Shortest Paths, *Conference Proceedings of 9th European Simulation Symposium*, pp. 89-93, Passau.

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

Tarapata Z. (1999a): Optimization of Many Task Sending in an Unreliable Parallel Computing System, *Proceedings of The 1st NATO Regional Conference on Military Communication and Information Systems*, October 6-8, Zegrze (Poland), vol. III, pp. 245-254.

Tarapata Z. (1999b): Simulation Method of Aiding and Estimation of Transportation Columns Movement Planning in Stochastic Environment, *Proceedings of the 13th European Simulation Multiconference*, The Society For Computer Simulation International, June 1-4, Warsaw, pp. 613-619.

Tarapata Z. (1999c): Metody wyznaczania i oceny planu przemieszczania obiektów w sieciach stochastycznych uwarunkowanych czasowo, *Sprawozdanie końcowe z realizacji pracy badawczej PBW 911*, Wojskowa Akademia Techniczna, Wydział Cybernetyki, Warszawa.

Tarapata Z. (2000a): A Concept of Parallel Algorithm for Determining Set of Disjoint Paths in a Network, *Proceedings of the 6th International Conference on Models and Methods in Automation and Robotics*, August 28-31, Międzyzdroje (Poland), vol. 2, pp. 895-902.

Tarapata Z. (2000b): Computer Simulation of Individual and Grouped Military Objects Redeployment, *Biuletyn Wojskowej Akademii Technicznej*, vol. 1, pp. 147-162.

Tarapata Z. (2000c): Computer Tool for Supporting and Evaluating Convoys Redeployment Planning, *Operations Research and Decision*, vol. 1, pp. 91-107.

Tarapata Z. (2000d): Modelling of Terrain for Necessities of Military Objects Movement Simulation, *Biuletyn Wojskowej Akademii Technicznej*, vol. 1, pp. 127-146.

Tarapata Z. (2000e): Multi-Paths Optimization in Unreliable Time-Dependent Networks, *Proceedings of The 2nd NATO Regional Conference on Military Communication and Information Systems*, 04-06 October, Zegrze (Poland), vol. I, pp. 181-189.

Tarapata Z. (2000f): Some Aspects of Multi-Convoy Redeployment Modelling and Simulation, *Proceedings of the 21st AFCEA Europe Symposium, Exposition*, Prague October 18-20 (CD publication).

Tarapata Z. (2001): Modelling, Optimisation and Simulation of Groups Movement According to Group Pattern in Multiresolution Terrain-Based Grid Network, *Proceedings of The 3rd NATO Regional Conference on Military Communication and Information Systems*, 10-12 October, Zegrze (Poland) , vol. I, pp. 241-251.

Tarapata Z. (2003a): Military Route Planning in Battlefield Simulation: Effectiveness Problems and Potential Solutions, *Journal of Telecommunications and Information Technology*, vol. 4, pp. 47-56.

Tarapata Z. (2003b): O problemie zliczania dróg w grafach, *Badania Operacyjne i Decyzje*, vol. 2, pp. 61-75.

Tarapata Z. (2004a): Decomposition Algorithm for Finding Shortest Paths in Grid Networks of Large Size, *Proceedings of the 15th International Conference on Systems Science*, September 7-10, Wrocław (Poland), vol. III, pp. 209-216.

Tarapata Z. (2004b): Modele i metody planowania i symulacji przemieszczania w systemie symulacyjnego wspomagania szkolenia operacyjnego, *Materiały XII Konferencji Naukowej „Automatyzacja Dowodzenia"*, 2-4 czerwca, Gdynia-Jurata.

Tarapata Z. (2004c): Models and Methods of Movement Planning and Simulation in Simulation Aided System for Operational Training, *Proceedings of The 6th NATO Regional Conference on Military Communication and Information Systems*, ISBN 83--920120-0-3, October 6-8, Zegrze (Poland), pp. 152-161.

Tarapata Z. (2005a): Planowanie i symulacja przemieszczania obiektów w symulacyjnych grach komputerowych – studium przypadku, *Materiały Konferencyjne z XI Warsztatów PTSK*, Białystok-Augustów, Warszawa, ISBN 83-88229-80-X, pp. 370-377.

Tarapata Z. (2005b): Synchronization Method of Many Objects Movement in Computer Generated Forces Systems, *Proceedings of The 7th NATO Regional Conference on Military Communication and Information Systems*, ISBN 83-92 0 120-3-8, October 4-5, Zegrze (Poland), pp. 93-99.

Tarapata Z. (2005c): Wielokryterialne problemy wyznaczania tras w sieciach komputerowych, w: Węgrzyn S., Czachórski T., Pochopień Cz. (red.): *Wysokowydajne sieci komputerowe. Nowe technologie*, Wydawnictwa Komunikacji i Łączności, Warszawa, pp. 183-193.

Tarapata Z. (2006a): Adaptacyjny algorytm wyznaczania tras z prognozowaniem obciążenia sieci, w: Grzywak A., Kwiecień A. , Klamka J., Pochopień Cz. (red.): *Nowe technologie sieci komputerowych, t.II*, Wydawnictwa Komunikacji i Łączności, Warszawa, pp. 31-41.

Tarapata Z. (2006b): Planowanie dróg bezkolizyjnych – metoda minimalizacji wpływu zagrożeń transportowych (katastrof drogowych i kolejowych), *opracowanie wewnętrzne*, Wydział Cybernetyki, WAT, Warszawa.

Tarapata Z. (2006c): Nieklasyczne modele i metody planowania tras w systemach wspomagania planowania ruchu: analiza złożoności, efektywności i zastosowań, *Logistyka*, vol. 6 (publikacja na CD).

Tarapata Z. (2007a): Modele harmonogramowania zsynchronizowanego przemieszczania wielu obiektów, *Badania Operacyjne i Decyzje*, vol. 2, pp. 83-103.

Tarapata Z. (2007b): Multicriteria Weighted Graphs Similarity and its Application for Decision Situation Pattern Matching Problem, *Proceedings of the 13th IEEE/IFAC International Conference on Methods and Models in Automation and Robotics*

(MMAR'2007), ISBN 978-83-751803-3-6, August 27-30, Szczecin, Poland, pp. 1149-1155.

Tarapata Z. (2007c): Nieklasyczne modele i metody planowania tras w systemach wspomagania planowania ruchu: analiza złożoności, efektywności i zastosowań, *Prace Naukowe Politechniki Warszawskiej,* seria Transport, Zeszyt 60, pp. 99-114.

Tarapata Z. (2007d): Selected Multicriteria Shortest Path Problems: An Analysis of Complexity, Models and Adaptation of Standard Algorithms, *International Journal of Applied Mathematics and Computer Science*, vol. 17(2), pp. 269-287.

Tarapata Z. (2007e): The Decision Automata for Manoeuvre Planning and Control in Simulation Aided System for Operational Training, *Proceedings of the Military Communications and Information Systems Conference MCC'2007*, ISBN 978-3-934401--16-7, September 25-26, Bonn, Germany.

Tarapata Z. (2008a): Algorytmy harmonogramowania zsynchronizowanego przemieszczania wielu obiektów, *Badania Operacyjne i Decyzje*, vol. 4, pp. 101-132.

Tarapata Z. (2008b): Automatization of Decision Processes in Conflict Situations: Modelling, Simulation and Optimization, in: Arreguin J.M.R. (edt.): *Automation and Robotics*, ISBN 978-3-902613-41-7, I-Tech Education and Publishing, Vienna (Austria), pp. 297-328.

Tarapata Z. (2008c): Modeling, Simulation and Optimization of Selected Decision Processes in Conflict Situations – A Case Study, *Polish Journal of Environmental Studies*, vol. 17(3B), pp. 467-474.

Tarapata Z. (2008d): Selected scheduling problems for synchronization of multi-objects movement, *Biuletyn Wojskowej Akademii Technicznej*, No.4 (652), vol. LVII, pp. 25-37.

Tarapata Z. (2008e): Zastosowanie metod wyznaczania przepływu w sieciach do planowania manewru wojsk, *Biuletyn Instytutu Systemów Informatycznych*, No. 2, pp. 31-44.

Tarapata Z. (2008f): Złożone niedeterministyczne problemy decyzyjne uwarunkowane czasowo w systemach zarządzania bezpieczeństwem narodowym, *Sprawozdanie końcowe z realizacji pracy badawczej PBW 571*, Wojskowa Akademia Techniczna, Wydział Cybernetyki, Warszawa.

Tarapata Z., Daleki Ł. (2008g): Wykorzystanie modeli pokrycia i alokacji zasobów do wspomagania decyzji w działaniach ratowniczych i reagowaniu kryzysowym, *Biuletyn Instytutu Systemów Informatycznych*, No. 2, pp. 45-58.

Tarapata Z. (2009a): Approximation Scheduling Algorithms for Solving Multi-Objects Movement Synchronization Problem, *ICANNGA'2009, Lecture Notes in Computer Science*, vol. 5495, Springer, Heidelberg, pp. 577-589.

Tarapata Z., Drozdowski T., Mierzejewski K. (2009b): Modele, metody i narzędzia wspomagania decyzji w sytuacjach zagrożenia systemu transportowego, *Przegląd Telekomunikacyjny i Wiadomości Telekomunikacyjne*, vol. 8-9, pp. 1256-1264.

Tarapata Z., Kasprzyk R. (2009c): An Application of Multicriteria Weighted Graph Similarity Method to Social Networks Analyzing, *Proceedings of the International Conference on Advances in Social Network Analysis and Mining*, July 20-22, Athens (Greece), IEEE Computer Society, ISBN 978-0-7695-3689-7, pp. 366-368.

Tarapata Z., Rulka J., Tarnawski T., Drozdowski T., Mierzejewski K. (2009d): Zagrożenia systemu transportowego, w: A.Najgebauer (red.) *Modele zagrożeń aglomeracji miejskiej wraz z systemem zarządzania kryzysowego na przykładzie m.st. Warszawy*, ISBN 978-83-61486-22-0, WAT, Warszawa, pp. 415-452.

Tarapata Z. (2010a): A Parallel Decomposition Algorithm for Shortest Path Problem in Large-Size Mesh Networks, *Biuletyn Wojskowej Akademii Technicznej*, vol. LIX, No. 3, pp. 295-306.

Tarapata Z. (2010b): Movement Simulation and Management of Cooperating Objects in CGF Systems: a Case Study, *KES-AMSTA'2010, Lecture Notes in Artificial Intelligence*, vol.6070, Springer, Heidelberg, pp. 293-304.

Tarapata Z. (2010c): Multiresolution Models and Algorithms of Movement Planning and their Application for Multiresolution Battlefield Simulation, *ACIIDS'2010, Lecture Notes in Artificial Intelligence*, vol. 5991, Springer, Heidelberg, pp. 378-389.

Tarapata Z., Chmielewski M., Kasprzyk R. (2010d): An Algorithmic Approach To Social Knowledge Processing And Reasoning Based On Graph Representation – A Case Study, *ACIIDS'2010, Lecture Notes in Artificial Intelligence*, vol. 5991, Springer, Heidelberg, pp. 93-104.

Tarapata Z., Kasprzyk R. (2010e): Graph-Based Optimization Method for Information Diffusion and Attack Durability in Networks, *RSCTC'2010, Lecture Notes in Artificial Intelligence*, vol.6086, Springer, Heidelberg, pp. 698-709.

Tarapata Z., Mierzejewski K. (2010f): Prognozowanie i symulacja skutków wystąpienia zagrożeń systemu komunikacyjnego aglomeracji, *Symulacja w badaniach i rozwoju*, vol. 1(1/2010), pp. 93-106.

Tarapata Z., Wrocławski S. (2010g): Subgraphs Generating Algorithm for Obtaining Set of Node-Disjoint Paths in Terrain-Based Mesh Graphs, *MIG'2010, Lecture Notes in Computer Science*, pp.6459, Springer, Heidelberg, pp.398-409.

Tarapata Z. (2010h): Integracja systemów dowodzenia, *Raport końcowy konsorcjanta z Wojskowej Akademii Technicznej z realizacji projektu rozwojowego nr MNiSW 0050/R/T00/2008/06*, Wydział Cybernetyki, Wojskowa Akademia Techniczna, Warszawa.

Tarapata Z. (2011a): Terrain-Based Modelling and Optimization of Groups Movement Using Group Patterns, *Biuletyn Instytutu Systemów Informatycznych*, No. 7 (in press).

Tarapata Z. (2011b): Movement Planning and Simulation of Military Units in "Zlocien" System: Models and Methods, *Biuletyn Instytutu Systemów Informatycznych*, No. 7, (in press).

Tarapata Z., Godlewski P. (2011c): Wielorozdzielcze modele i algorytmy planowania przemieszczania oraz ich zastosowanie w wielorozdzielczej symulacji pola walki, *Symulacja w badaniach i rozwoju*, vol. 3 (in press).

Tarapata Z., Wrocławski S. (2011d): Metody rozwiązywania problemu najkrótszych dróg wierzchołkowo rozłącznych przechodzących przez wybrane wierzchołki w sieciach o strukturze kraty, *Biuletyn Wojskowej Akademii Technicznej*, vol. 3 (in press).

Tarjan R.E. (1983): Data Structures and Network Algorithms, *Society for Industrial and Applied Mathematics*, Philadelphia, Pennsylvania.

Tsaggouris G., Zaroliagis Ch. (2005): Improved FPTAS for Multiobjective Shortest Paths with Applications, *Technical Report No. TR 2005/07/03*, Research Academic Computer Technology Institute.

Tuft D., Gayle R., Salomon B., Govindaraju N., Lin M., Manocha D. (2006): Accelerating Route Planning and Collision Detection for Computer Generated Forces Using GPUS, *Proc. of Army Science Conference*, Orlando (USA).

Tung C.T. Chew , K.L. (1992): A Multicriteria Pareto-Optimal Path Algorithm, *European Journal of Operational Research*, vol. 62, pp. 203-209.

Tung C.T., Chew K.L. (1988): A Bicriterion Pareto-Optimal Path Algorithm, *Asia-Pacific Journal of Operational Research*, vol. 5, pp. 166-172.

Umeyama S. (1988): An Eigen Decomposition Approach to Weighted Graph Matching Problems, *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 10(5), pp. 695-703.

Undeger C., Polat F., Ipekkan Z. (2001): Real-Time Edge Follow: A New Paradigm to Real-Time Path Search, *Annual European Game-On Conference*, London, UK, November 30-December 1.

Urrutia J. (1999): Handbook of Computational Geometry, *Elsevier Science Publishers*, B.V. North-Holland, Amsterdam.

Vassilvitskii S., Yannakakis M. (2004): Efficiently Computing Sufficient Trade-off Curves, in Automata, Languages, and Programming, *ICALP'2004, Lecture Notes in Computer Science*, vol. 3142, Springer, Heidelberg, pp. 1201-1213.

Van der Akker, M., Geraerts R., Hoogeveen H, Prins C. (2010): Path Planning for Groups Using Column Generation, *MIG'2010, Lecture Notes in Computer Science*, pp. 6459, Springer, Heidelberg, pp. 94-105.

Wang X., Chen G. (2003): Complex Networks: Small-World, Scale-Free and Beyond, *IEEE Circuits and Systems Magazine*, vol. 3 (1), pp. 6-20.

Wang, Y. (2006): Numerical Modelling of Autonomous Agent Movement and Conflict, *Computational Management Science*, vol. 3(3), pp. 207-223.

Weng M, Wei X., Qu R., Cai Z. (2009): A Path Planning Algorithm Based on Typical Case Reasoning, *Geo-spatial Information Science*, vol. 12(1), pp. 66-71.

Warburton A. (1987): Approximation of Pareto Optima in Multiple-Objective, Shortest-Path Problems, *Operations Research*, vol. 35, pp. 70-79.

Watts D. J., Strogatz S. H. (1998): Collective dynamics of „small-world" networks, *Nature*, vol. 393, pp. 440-442.

Wellman M., Ford M., Larson K. (1995): Path Planning under Time-Dependent Uncertainty, *Proceedings of the 11th Conference of Uncertainty in artificial intelligence*, Montreal, pp. 532-539.

White D.J. (1987): The Set of Efficient Solutions for Multiple Objective Shortest Path Problems, *Computers and Operations Research*, vol.9(2), pp.101-107.

Wijeratne A.B., Turnquist M.A., Mirchandani P.B. (1993): Multiobjective Routing of Hazardous Materials in Stochastic Networks, *European Journal of Operational Research*, vol. 65, pp. 33-43.

Wilson R.J. (1998): Wprowadzenie do teorii grafów, *PWN*, Warszawa.

Wu Q., Hartley J., Al-Dabass D. (2005): Time-Dependent Stochastic Shortest Path(s) Algorithms for a Scheduled Transportation Network, *Intern. Journal of Simulation*, vol. 6 (7-8), pp. 53-60.

Zafar, K., Qazi, S.B., Baig, A.R. (2006): Mine Detection and Route Planning in Military Warfare using Multi Agent System, in: *Proceedings of the 30th Annual International Computer Software and Applications Conference COMPSAC'*2006, Chicago, vol. 2, pp. 327-332.

Zeigler B., Rozenblit J., Chtistensen E. (1991): Reducing the Validation Bottleneck with a Knowledge-Based, Distributed Simulation Environment, *Expert Systems with Applications*, vol. 3, Issue 3, pp. 329-342.

Zeigler B., Cho T., Rozenblit J. (1996): A Knowledge-Based Simulation Environment for Hierarchical Flexible Manufacturing, *IEEE Transactions On Systems, Man, And Cybernetics-Part A: Systems And Humans*, vol. 26(1), pp. 81-90.

Zhan F. B., Noon C. E. (1998): Shortest Path Algorithm: An Evaluation using Real Road Networks, *Transportation Science* , vol. 32, pp. 65-73.

Zhan F. B., Noon C. E. (2000): A Comparison Between Label-Setting and Label-Correcting Algorithms for Computing One-to-One Shortest Paths, *Journal of Geographic Information and Decision Analysis*, vol. 4, pp. 1-13.

Zhao Y. (1997): Vehicle Location and Navigation Systems, Artech House Publishers.

Zlocien (2002): *Projekt techniczny systemu symulacyjnego wspomagania szkolenia operacyjnego pk. Złocień,* t. II, Wydział Cybernetyki, WAT, Warszawa.