8. Социальные сети в современном рекрутинге // Электронный научный журнал факультета жерналистики МГУ им. М.В. Ломоносова [Электронный ресурс]. – Режим доступа: http://www.mediascope.ru. – Дата доступа: 20.12.2014.
9. Батура, Т.А. Методы анализа компьютерных социальных сетей / Т.А. Батура // Обзор работ, посвященных проблеме анализа компьютерных социальных сетей. Институт систем информатики им. А.П. Ершова СО РАН, Новосибирск.
10. Social Network Sites: A Definition // Nicole B. Ellison Social Network Sites: Definition, History, and Scholarship. // Journal of Computer-Mediated Communication. 2007. [Electronic resource]. – Mode of access: http://onlinelibrary.wiley.com. – Date of access: 20.12.2014.
11. Robert, B. Doorenbos Production Matching for Large Learning Systems / B. Robert // Computer Science Department. Carnegie Mellon University. – Pittsburgh, PA. 1995.

UDC 519.854

SOFTWARE IMPLEMENTATION OF DIJKSTRA'S ALGORITHM

NICOLAI GURTOVENKO, OKSANA GOLUBEVA
Polotsk State University, Belarus

Currently, there are many algorithms to find the shortest way. The most effective of them is Dijkstra's algorithm. This article is devoted to the software implementation of this algorithm and its interface design, convenient for the use of schoolchildren, students and teachers in order to quickly resolve the problem of finding the shortest way.

Navigating troubled people has long been a problem. Navigation history begins from the time of trade caravans, the development of relations between nations, military campaigns. Even at those times rough maps and routes were drawn. Navigation continued to evolve. Later, travelers started to draw maps of the whole world; maps of individual regions appeared. In the XX century science and industry began to develop actively, and it led to the emergence of artificial satellites and allowed to draw a detailed map of the earth. The most recent inventions are navigators that help a person to move in the direction of a certain point, not knowing the exact route, using the communication with the satellite. Navigators are now built into all smartphones and offer a variety of programs, allowing to determine your location and build up the desired route. Also there is a possibility of communication between users with the help of navigators, tracking traffic jams, speed, accidents and even traffic police posts. The most famous ones are «Yandex Navigator», «Navitel», «OsmAnd» and others.

Roads are a network. A network is a connected digraph without loops, the weight of each arc in which is a natural number (the capacity of the arc). The shortest path is a path with the lowest cost in passing (financial, fuel, time, ect.).

One of the algorithms for finding the shortest way is Dijkstra's algorithm. The algorithm was invented by a Dutch scientist E. Dijkstra in 1959 and is today considered one of the most efficient algorithms for finding the shortest way. The algorithm works with networks without negative weight edges (if we have a one-way road it will not take into account going in the opposite direction; it will not let you go along the opposite lane). In terms of software implementation Dijkstra's algorithm is quite simple. It needs reasonable system resources increasing the speed of the construction of the way. One disadvantage is the fact that it is not free. The algorithm is patented and its commercial use is not free of charge.

An overview of analogs of the algorithm

There are several analogs of Dijkstra's algorithm. The most popular ones are:

– Bellman-Ford's algorithm finds the shortest way from one vertex of the graph to all others in a weighted graph. Weight edges can be negative.

– The A* search algorithm finds the least wasteful route from one vertex (primary) to another (target, final) using a searching algorithm based on the first best match on the graph.

– Floyd – Warshall's algorithm finds the shortest way between all nodes of a directed weighted graph.

– Johnson's algorithm finds the shortest ways between all pairs of vertices of a directed weighted graph.

– Lee's algorithm (wave algorithm) is based on the method of widthway search. It finds the way between the vertices of the graph of **s** and **t** (**s** doesn't match **t**), contains a minimum number of intermediate vertices (ribs). Its main application is tracing the electrical connections on the crystals and chips on printed circuit boards. It is also used to find the shortest distance on the map in strategic games.

– Kildall's algorithm also finds the shortest way.

– Kosaraju's algorithm is used for finding ways in directed graphs.

All of the algorithms listed above can be used to find the shortest way, but some of them have found another use. Below we will describe each algorithm in more detail.

Bellman-Ford's algorithm is the algorithm to find the shortest way in a weighted graph. Over time *O* (|*V*| × |*E*|) algorithm finds the shortest way from one vertex of the graph to all others. Unlike Dijkstra's algorithm, Bellman-Ford's algorithm admits edges with negative weights. It was proposed independently by Richard Bellman and Lester Ford.

The A* search algorithm in computer science and mathematics uses a best-first search and finds a least-cost way from a given initial node to one goal node (out of one or more possible goals). As A* traverses the graph, it follows a way of the lowest expected total cost or distance, keeping a sorted priority queue of alternate way segments along the way. It uses a knowledge-plus-heuristic cost function of node x (usually denoted *f*(*x*)) to determine the order in which the search visits nodes in the tree. The cost function is a sum of two functions:

– the past path-cost function, which is the known distance from the starting node to the current node *x* (usually denoted *g*(*x*));

– the future path-cost function, which is an admissible "heuristic estimate" of the distance from *x* to the goal (usually denoted *h*(*x*)).

The *h*(*x*) part of the *f*(*x*) function must be an admissible heuristic; that is, it must not overestimate the distance to the goal. Thus, for an application like routing, *h*(*x*) might represent the straight-line distance to the goal, since that is physically the smallest possible distance between any two points or nodes.

Floyd-Warshall's algorithm is a graph analysis algorithm for finding shortest ways in a weighted graph with positive or negative edge weights (but with no negative cycles) and also for finding transitive closure of a relation *R*. A single execution of the algorithm will find the lengths (summed weights) of the shortest ways between all pairs of vertices, though it does not return details of the ways themselves.

Johnson's algorithm is a way to find the shortest ways between all pairs of vertices in a sparse, edge weighted, directed graph. It allows some of the edge weights to be negative numbers, but no negative-weight cycles may exist. It works by using the Bellman–Ford algorithm to compute a transformation of the input graph that removes all negative weights, allowing Dijkstra's algorithm to be used on the transformed graph

If Dijkstra's nondecreasing priority queue is implemented as a Fibonacci heap, then Johnson's algorithm work is still *O* (*V* ^ 2 \ log *V* + *VE*). A more simple implementation of a non-decreasing priority queue time work becomes *O* (*VE* \ log *V*), but for sparse graphs this value in the asymptotic limit behaves better than the running time of Floyd-Warshall's algorithm.

Lee's algorithm (wave algorithm) is used in the development of printed circuit boards and is widely distributed in computer games. The problem of finding the shortest way between points A and B in the field of play with randomly placed obstacles is characterized, mostly for today's popular tactical and strategic games. As a secondary problem, it can occur in almost any games such as RPG, quests, logical games (a typical example is "Color Lines"). Why is it necessary to seek the shortest route? In some games, such as "UFO-2", "Laser Squad", the length of the route depends on the number of units of time spent. It means that the more optimal way will be found, the faster the warrior gets to the goal. But, for example, in "Color Lines", the path length is not stipulated by the rules; the only significant fact is the possibility or impossibility of moving the ball. But in this game it will be nice, if the ball goes straight to where it is shot not defiling mysteriously across the game board. The solution to this problem comes to us from a very remote area such as electronics, to be exact PCB laying out.

Kosaraju's algorithm works as follows. Let *G* be a directed graph and *S* be an empty stack. While *S* does not contain all vertices, choose an arbitrary vertex *V* not in *S*. Perform a depth-first search starting at *V*. Each time that depth-first search finishes expanding a vertex *U*, push *U* onto *S*. Reverse the directions of all arcs to obtain the transpose graph. While *S* is nonempty, pop the top vertex *V* from *S*. Perform a depth-first search starting at *V* in the transpose graph. The set of visited vertices will give the strongly connected component containing *V*; record this and remove all these vertices from the graph *G* and the stack *S*. Equivalently, breadth-first search (BFS) can be used instead of depth-first search.

Statement of the problem and the implementation of the algorithm

When writing a program for Dijkstra's algorithm implementing, we were faced with certain problems. We were looking for a prefabricated version of its implementation. But among open source software we have not found anything worthwhile but for non-working pieces of code. If we talk about finished assemblies, we have found one program working on the algorithm. However, it is impossible to give a matrix of weights there directly, only through drawing a network. Due to the lack of ready code, we wrote it ourselves using the search engine «Google» looking for certain functions and exploring some obscure points. While searching an online service was found to solve problems of Dijkstra's algorithm. Solutions are given there in detail, the matrix is set conveniently. The only drawback is the inability to work offline and no graphic representation.

At the very start a rough outline of the program was written, the so-called "pseudo", including key functions with their responsibilities and interrelations. However, in practice it turned out to be more complicated. Implementing of the algorithm itself was easy, knowing its mathematical representation. It was necessary to simply "be translated into a programming language." At this stage, the program is a console application in which you want to set a matrix of weights (the program asks for a specific value, the user enters it), and then a starting and final points are requested and the shortest way is calculated. In the future, we plan to issue a more intuitive and simple interface, add a graphical representation of the network, add some modifications of the algorithm.

The program will be useful for pupils and students in order to understand Dijkstra's algorithm while dealing with the problem of finding the shortest way. The program is also useful for teachers to test the solutions pupils and students find quicker. Developing this project, you can get quite an interesting, useful product, besides being unique and versatile.

UDC 004.896:613.62

APPLICATION OF FUZZY LOGIC IN MODEL OF OCCUPATIONAL RISK ASSESSMENT

ALENA HALYNSKAYA, YULIYA BULAUKA
Polotsk State University, Belarus

The paper discusses the use of fuzzy logic in the problem of occupational risk assessment.

Soft computing includes fuzzy logic, neural networks, probabilistic reasoning, and genetic algorithms. Today, techniques or a combination of techniques from all these areas are used to design an intelligence system. Neural networks provide algorithms for learning, classification, and optimization, whereas fuzzy logic deals with issues such as forming impressions and reasoning on a semantic or linguistic level.

Fuzzy logic was initiated in 1965 [1] by Lotfi A. Zadeh, professor for computer science at the University of California in Berkeley. Basically, fuzzy logic is a multivalued logic that allows intermediate values to be defined between conventional evaluations like true/false, yes/no, high/low, etc. Notions like rather tall or very fast can be formulated mathematically and processed by computers, in order to apply a more human-like way of thinking in the programming of computers [2].

In 1993 Kosko (Kosko) proved a theorem on fuzzy approximation (FAT – Fuzzy Approximation Theorem) [3], which states that any mathematical system can be approximated by a system of fuzzy logic. Therefore, using natural language rules "If – then" followed by their formalization by means of the theory of fuzzy sets can be any arbitrary accurately reflect the relationship "Input Output" without the use of complex apparatus of differential and integral calculus, traditionally used in the management and identification.

Fuzzy logic has emerged as a profitable tool for the controlling and steering of systems and complex industrial processes, as well as for household and entertainment electronics, as well as for other expert systems and applications like occupational risk assessment.

In the real world, vagueness and ambiguity exist because of the limitations of our language and other factors, such as context and perception. Closely related to this ambiguity is the question of lexical imprecision in natural language; when expressing knowledge, individuals would rather use words than numbers.

Occupational risk assessment deals with uncertain situations, that is, situations in which we do not have complete and accurate knowledge about the system state, such as estimate severity consequences of occupational accidents.

Additionally, legal records, statistical data and site documentation produced by companies are generally insufficient for determining risks. On-site inspections generally use linguistic expressions rather than metrics to assess the safety risks. These facts increase the imprecision and inaccuracies of the occupational risk assessment process, and this imprecision is the reason why we use a fuzzy approach.

For systems in which imprecise and inaccurate information is available, fuzzy concepts and techniques provide suitable ways to collect observed input data and represent it in a uniform and scalable way. Fuzzy sets seem to be quite relevant in three classes of applications: classification and data analysis, reasoning under uncertainty, and decision-making problems.

In our work, we use the lattermost application of decision making because it will allow the combination of all risk factors using aggregation operators to define a general level of risk assessment.

A fuzzy inference system (FIS) essentially defines a nonlinear mapping of the input data vector into a scalar output, using fuzzy rules. The mapping process involves input/output membership functions, FL operators, fuzzy if – then rules, aggregation of output sets, and defuzzification.

The FIS contains four components: the fuzzifier, inference engine, rule base, and defuzzifier. The rule base contains linguistic rules that are provided by experts. It is also possible to extract rules from numeric data. Once the rules have been established, the FIS can be viewed as a system that maps an input vector to an output vector. The fuzzifier maps input numbers into corresponding fuzzy memberships. This is required in order to