View metadata, citation and similar papers at core.ac.uk
brought to you by CORE

PROFESSIONAL EDUCATION

UDC 519.101:656.022
DOI: 10.18372/2306-1472.72.11989

Yevgeny Gayev[1]
Vadim Kalmikov[2]

THE TRAVELLING SALESMAN PROBLEM IN THE ENGINEERING EDUCATION PROGRAMMING CURRICULUM

[1,2]National Aviation University
1, Kosmonavta Komarova prospect, 03680, Kyiv, Ukraine
E-mails: [1]Ye_Gayev@voliacable.com, [2]kvvad@ukr.net

Abstract

Objective*:** *To make students familiar with the famous Traveling Salesman Problem (TSP) and suggest the latter to become a common exercise in engineering programming curriculum provided the students master computer science in the easy programming environment MATLAB.* ***Methods: *easy programming in MATLAB makes true such modern educational approach as "discovery based" methodology.* ***Results:*** *a MATLAB TSP-program oriented to Ukrainian map is suggested that allows to pictorially demonstrate the process of optimal route search with an option to decelerate or accelerate the demonstration. The program is guessed to be useful both for learning the TSP as one of fundamental logistics problems and as an intriguing programming curriculum excersize. Several sub-programs according to key stone Computer Science Curriculum have also been suggested. This lies in line with recent "discovery based" learning methodology.* ***Discussion:*** *we explain how to create this program for visual discrete optimization, suggest required subprograms belonging to key stone programming algorithms including rather modern graphical user interface (GUI), how to use this MATLAB TSP-program for demonstration the drastical grows of solution time required.* ***Conclusions:*** *easy programming being realized in MATLAB makes dificult curriculum problems attractive to students; it focuses them to main problem' features, laws and algorithms implementing the "discovery based" methodology in such a way.*

Keywords: combinatorics; discovery based learning methodology; logistics; optimization; programming; programming education; TSP (Traveling Salesman Problem).

1. Research motivation

Recent role of the Computer Science in the engineering education curriculum is very serious, as it effects all the educational process for today's students. Indeed, Computer Science, and particularly Programming as its sub-discipline, provides to students not only a crucial working instrument but influences effectiveness of the whole educational process as well, as the first author stated in [1-5].

Programming teacher or that of Computer Science at all can of course follow traditional way by teaching a number of particular high-level languages like Pascal and Delphi, C and C^{++}, Java etc. However, many teachers have been looking for a long time for more effective methods to teach programming to students ranging from beginners to advanced, from public schools to universities [6]. Historically the first were the languages BASIC (abbreviation of Beginner's All-purpose Symbolic Instruction Code) and Logo (1960-ies), ABC (1987) [6-8], two Russian projects Robic (Робик) and DRAGON (ДРАКОН, 2011) [9], Scratch (2013) and others [6-8].

Bearing in mind the above historical experience, the first author has gathered an extended experience with teaching "Programming with MATLAB" for first year university students, and, along with it, would like to background such an approach by means of further collecting educational texts and exercises [1-6]. Below is one more example to apply

an easy MATLAB programming to very difficult classical problem.

The famous Travelling Salesman Problem (TSP) turned to be one of hardest in the discrete mathematics, and thus had not been included to any programming courses for beginners [10]. Meanwhile, the TSP is significant to a series of engineering specialties like aero navigation, logistics, road planning, manufacturing microchips, discrete mathematics etc. It were thus worthwhile to include it to computer science discipline provided the latter disposes a platform to treat the TSP sufficiently easy. Such platform is MATLAB.

2. Problem formulation. State-of-the-Art

Mathematical formulation of the TSP sounds in the following way: a "salesman" from the city 0 should visit n cities named as 1, 2, . . . , n only once and come back to 0, and among $N=n!$ possible routes the shortest is to be found.

The early origins of the TSP are unclear [11]. A handbook for travelling salesman from 1832 mentioned the problem and included example tours through Germany and Switzerland but contained no any mathematical treatment [11-13]. The TSP was defined strongly in the 1800s by mathematicians W.R.Hamilton and Th.Kirkman. Later in 1930s, the TSP became studied by mathematicians from Vienna and Harvard, notably by K.Menger. They applied the obvious brute-force algorithm (applied here as well) and started development of a variety of more sophisticated approaches based on graph theory and modern algebra. Up-to-date approaches do also use heuristic and neural net approaches.

During last decades, the TSP has found a number of further practical applications. There are GPS navigators [14], logistics, aero navigation etc. among them. Similar applications have also appeared where the concept of cities was replaced, for example, with that of customers, soldering points, or DeoxyriboNucleic Acid (DNA) fragments, and the measure of distance was replaced with travelling times or costs, or, similarly, with a measure how DNA fragments are close to one another. Despite its practical importance and the long history, the TSP still remains unsolved "in full". In many cases, additional constraints are to be accounted for, such as limited resources or time, landscape orography etc. that make the problem considerably harder. The problem's state of the art has been completely described in popular resources as [11,13].

3. Aim of the work

We do not pretend to make any original discovery in the TSP. We do pretend only to make this problem available to students in their mastering the programming course [10]. In the perspective, they may wish to develop any of recent effective computational methods mentioned over. We need, however, to start from the easiest one.

Direct solution of the TSP lies in the "honest" search of all the possible routes from the city 0 through all the others, accurate calculation of their lengths and comparing them until all the variants will be examined. This "naive" and direct approach is called the "brute-force algorithm" in the literature [11,12]. Existence of a solution, i.e. existence of a minimum among big but finite number $N=n!$ of route lengths, is evident. However, it may be not unique. Discussion about the time consumed by the algorithm we postpone until the section 5.

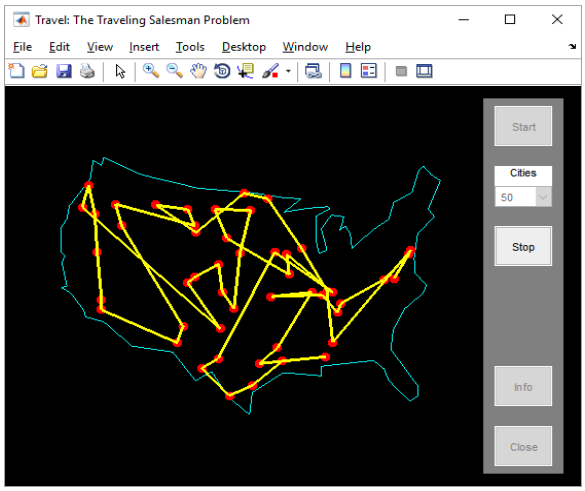

Fig. 1. View of common MATLAB TSP-program

Having chosen the method, we aim to create a MATLAB program for TSP to be easy to students to understand and to reproduce. Actually, the Traveling Salesman Problem has already been realized in MATLAB. One should simply run in the Command Window its internal command

$$>> \textbf{\textit{travel}} \qquad (1)$$

(the symbol >>, the *prompt*, denotes relation to the Command Window of MATLAB here and below). Graphical User Interface (GUI) that appears is shown in the Figure 1 along with solution for n=50 cities on American map. It is imperfection of this program that

enumeration of routes currently in consideration is too fast to follow them and to understand the process. Despite the code of this program is open, it is difficult to make students to reproduce a program like this of their own.

We intend (*i*) to elaborate for students such a practical task on their major in aero navigation disciplines, (*ii*) provide them required knowledge accordingly to the Computer Science Curriculum [10], (*iii*) to encourage them to their original research and development, and (*iv*) to allow research of their own by means of programming, at least testing the program efficiency (section 5). It is to account as well that our program is to be oriented to our country Ukraine rather than to USA. However, any map may be used.

It is our intention in this problem to involve important educational curriculum material [11] in our program elaboration, i.e. to develop our own programs rather than to use ready ones. Educational approach described lies in line with the World educational concept of Discovery Based Learning [7,8].

4. Program development

We plan our program in the following way. First, one needs to create Graphical User Interface (GUI) of the program where all its functionality should be foreseen by means of several windows for inputting and outputting information, by buttons that provide other information required and start execution of certain particular tasks (GUI elements). One of latter is, for example, a procedure how to choose particular cities. Secondly, a subprogram is to be elaborated that enumerates all possible routes between the cities chosen. Thirdly, all the routes generated are to be "passed by the program" (the brute-force method) and the length of them calculated. It would be visual to demonstrate on the map each route currently in consideration having controlling duration between routes. By setting artificial pauses between routes to zero, we accelerate calculation process to its minima; setting them to 1 or 2 seconds, we provide a time to user for analyzing current stage of the search. Finally, the route with the least length is to be returned as the final problem solution for the city collection chosen. The uniqueness of the route is not guaranteed, however.

To realize this plan, the following development stages were performed.

4.1. GUI with the map of Ukraine. First, a Graphical User Interface is to be created by means of standard MATLAB' environment that appears after issuing the command

$$>> \textbf{\textit{guide}} \; . \qquad (2)$$

Such portion of the whole development has already been described in [2,4,5]. Position and size of the buttons and windows are designed at this stage, particularly graphical window (called axis) along with their colors, text strings, their fonts and other parameters through MATLAB's Property Inspector. It is important to mention to set simultaneously *handle names* (called *pointers* or *references* in other computer languages) for all of them GUI elements to control their correspondent properties.

One could project appropriate background *jpg*-picture to the whole GUI (use *imagesc* for this) or/and display a geographical map taken as a map of Ukraine to axis at this stage, position 1 in Fig. 2. MATLAB-commands *imread* and *imshow* are used for this.

All other functionality, elements 2 – 7 are to be also foreseen at this stage. Say, the button 2 should provide brief explanation how to use and to run this GUI. For this, one could use the ready GUI-command *helpdlg*('*Help text*', '*GUI header*'), Fig.3. The latter should be associated with the button 2 by means of a simple function explained in 4.6.

4.2. Choosing cities from the map. Realizing the functionality of the program, the number of cities *M* that foreseen on the map should be chosen from the window 3 first, and then set the timing between routes, window 4, Fig. 2. City Kyiv is taken as a *StartingCity* №0, and *M* other cities will be chosen from a *CityCollection* file. Two MATLAB's data types may be utilized for the *CityCollection* to keep them in computer memory with their map coordinates, either *cell* or *struct*. We have chosen the latter in this work. The structure *CityCollection* saves so the following information, both textual and numerical arrays, in its fields **.***City*, **.***X* and **.***Y*

$$\textit{CityCollection}\textbf{.}\textit{City}=[\text{'Kyiv'}; \text{'Lviv'}; \ldots],$$
$$\textit{CityCollection}\textbf{.}\textit{X}= [X_{\text{Kyiv}}, X_{\text{Lviv}}, \ldots],$$
$$\textit{CityCollection}\textbf{.}\textit{Y}= [Y_{\text{Kyiv}}, Y_{\text{Lviv}}, \ldots]$$

– *M* elements in each, where 'City', X_{City} and Y_{City} are names and coordinates of corresponding city on

the map expressed in *kilometers*. The cites chosen will get their numbers 1, 2, etc..

4.3. Generating TSP task consists of two operations. First, the number of cities is to be chosen from the *Listbox* or *Pop-up-Menu* 3, Fig. 2. Then, the *Pushbutton* 5 "Generate cities" is to be pressed. Particular cities from the *CityCollection* are to be chosen that form a numeric matrix *CitiesChosen* with the dimensions *M* to 3. In it, the first column *CitiesChosen*(:,1) contains numbers of chosen cities in the *CityCollection* data base, the second column *CitiesChosen*(:,2) saves their *X*-coordinates and the third one *CitiesChosen*(:,3) their *Y*-coordinates on the map expressed in *kilometers*. So the task for the Traveling Salesman Problem has been finally formed. When chosen, corresponded cities become labeled as green points on the map. The following plotting command may be used for this:

plot(*CitiesChosen*(:,2),*CitiesChosen*(:,3), 'go') (3)

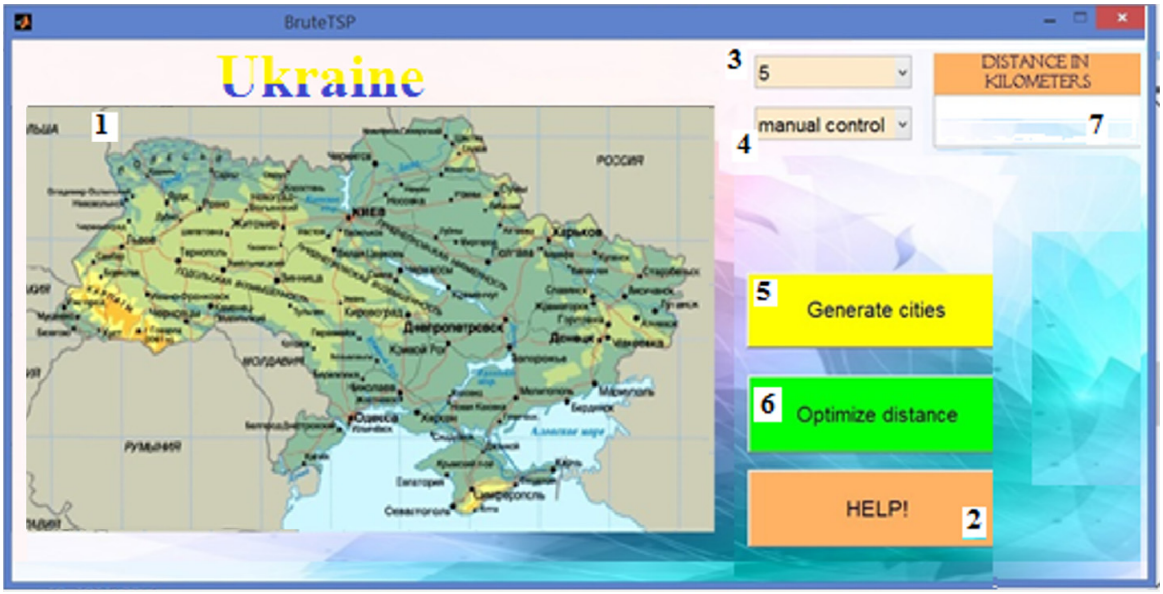

Fig. 2. GUI of MATLAB program for solving TSP up to 20 cities: 1 – map of the country; 2 – button for getting Help; 3 – *Listbox* window for entering number of cities to find least route; 4 – setting artificial time delay; 5 – *Pushbutton* for picturing cities on the map; 6 – *Pushbutton* for starting least route search; 7 – display of the least route length found

Note that the *StartingCity* should be included in this Task List as the first and the last elements.

4.4. Recursive generator of permutations. Now, the full enumeration of all possible routes through these *M* cities is to be prepared, the quantity of those is $M!/2$ if do not account the same but reverse (anti-symmetrical) routes. There is a command *perms* in MATLAB, but recall that our educational purpose is to teach students to develop programs of their own. That is why we would like to suggest such.

Solution suggested below is based on recursion what is one of key stone algorithms in Computer Science Curriculum [10]. Say, we need to get all permutations of numbers 1, 2, . ., *M*, what means that function *P*=*MyPermutations*(*M*) depended on *M* is to produce a matrix *P* where rows contain all permutations of numbers from 1 to *M*. In the simplest case *M*=1 the list of permutations *P* contains only 1. If *M*=2 the *MyPermutations*-program returns *P* as a matrix [1, 2] with R_2=1 rows (index stands for argument *M*=2). The latter contains three "empty positions": at the front, at the end and between all present elements. That is why *MyPermutations*(3)=[1 2 3; 1 3 2; 3 1 2] contains R_3=3 rows each created by putting the new element 3 to "empty positions" (note, reverse routes like [3 2 1; 2 3 1; 2 1 3] have not been accounted for). It means, similarly, that the matrix P_M=*MyPermutations*(*M*) may be obtained from that of P_{M-1} by the following algorithm: *M* new rows is to be obtained from each row of P_{M-1} by inserting new element numbered *M*. The number of matrix rows grows to P_{M-1}*P_M. Each row presents possible Salesman route. Such algorithm is realized in the next program that uses MATLAB's matrix technique:

Listing *MyPermutations.m* .

```
function P=MyPermutations(n)
%Returns matrix of all the Permutations of 1, 2, 3, ..., n
%without anti-symmetrical ones.
%It bases on the recursion [2,10].
%How to use:
%>> P5=MyPermutations(5),   etc.
%Copyright Gayev-Kalmykov, 2016.

if n = = 1
    P=1;
elseif n = = 2
    P=[1, 2]; % Permutations in these cases M=2
else
   %Previous Permutations with M1 rows, (n-1) columns:
    P1=MyPermutations(n-1);
    Sz=size(P1); N1=Sz(1); %size of the P1
    Ind1=0;  % Ind1 counts Rows in the future P
    for i=1:N1 %Row by Row of the P1
        Ind1= Ind1+1;
        RowP1=P1(i,:); %the P1 Row № i
       %first of all these N new Rows:
        P(Ind1, :)=[n, RowP1];
        for j=1 : n-1
            Ind1=Ind1+1;
            Row1=RowP1(1 : j);
            Row2=RowP1(j+1 : end);
            P(Ind1,:)=[Row1, n, Row2];
        end
    end
end
```

MATLAB-comments supplement code elements (the tests after %). This program runs when the button 5 is pressed. It returns the matrix with enumeration of all possible routes between chosen cities.

4.5. "Brute-force" process, the final action, is actually very simple: a program associated with the button 6 "Optimize distance", Fig. 2, searches all the $\frac{1}{2}M!$ routes represented by the formal argument n. A route for each permutation, say [7 5 12 . . .], consists of city names

CityCollection.*City*(0),
CityCollection.*City*(7),
CityCollection.*City*(5),
CityCollection.*City*(12),
. . .,
CityCollection.*City*(0)

and their associated X-coordinates

CityCollection.*X*(0),
CityCollection.*X*(7),
CityCollection.*X*(5),
CityCollection.*X*(12),
. . .,
CityCollection.*X*(0)

(4)

as well as Y-coordinates

CityCollection.*Y*(0),
CityCollection.*Y*(7),
CityCollection.*Y*(5),
CityCollection.*Y*(12),
. . .,
CityCollection.*Y*(0).

This route [7 5 12 . . .] is plotted with red lines in the graphical window 1 by the plotting command like (3) over the map picture. Then the calculation of the whole length of the current route is performed and the result displayed in the window 7, Fig. 2. Time delay *pause*(*dT*) is made before examination of the next route. The latter time delay *dT* is to be specified in the window 4. Figure 4 demonstrates each transitional route on the map with its current length for the TSP task of four cities. After this delay, the next route will be examined until the last one generated by the procedure *MyPermutations*.

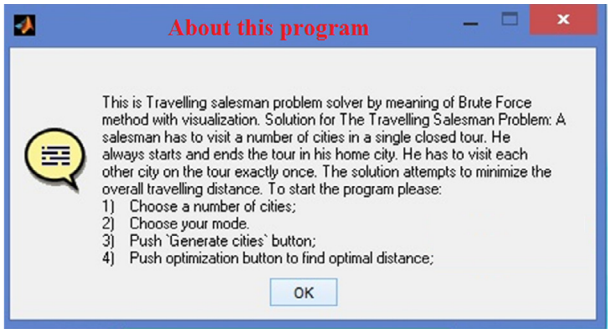

Fig. 3. View of the Help window

4.6. Final program view. We described all the functionality that we foreseen in the graphical program displayed in the Fig. 2. Resulting GUI developed in GUI-editor (2) is saved in MATLAB workspace in two files with the same name say *BruteTSP*, and with two extensions **.***fig* and **.***m*. The first one is a binary file keeping all the visual information (GUI element positions, their colors, text strings, fonts etc.) but the second is the text with the future code. The latter includes several function signatures that contain *_Callback* suffix in their names. All they correspond to particular GUI-elements in the whole GUI-program in the Fig. 2. In each of these GUI-associated functions, empty yet,

certain portions of programming code is to be written. Let us shortly overview them.

Each 'event', i.e. use of the corresponded GUI-element, should lead to execution of certain code portion that returns particular values into the whole process. In the case of *Listbox* 4, for example, the following code is to be written to *m*-file accordingly to automatically generated hint:

contents = *cellstr*(*get*(*hObject*, 'String'));
dT=*contents*{ *get*(*hObject*,'Value')};

The Value=5 chosen in *Listbox* 3 (Fig. 2) is also assigned to *M* in such a way.

However, all the data produced in each function is to be made available to other functions by placing them into mutual memory areas. For this, the code fragments

global M_cities

and

global dT

are to be present in functions associated with GUI-elements 3 and 4, Fig. 2.

Then, the *Pushbutton* 5 is to be pressed. To perform actions described in 4.2 – 4.4, the code of associated _Callback-function should get the above mentioned data for what its code is to begin with the string

global M_cities CitiesChosen P (5)

The last data will be produced here when the 'event' "Generate cities" happens. For this, the actions described in 4.2 and 4.3 including plotting (3) are to be performed. Then permutation generator described in 4.4 is run. It produces permutations *P*, i.e. all the possible roots, that are, again, placed into the global memory area by (5).

Now is the turn of the last *Pushbutton* 6, "Optimize distance". Pressing it activates algorithm described in the section 4.5. The code of corresponded *_Callback*-function begins with

global M_cities dT CitiesChosen P

Each of permutations *P* are checked out one-by-one. "The salesman" travels the each route (4) indicated by *P*, this route is plotted as (3), the route length is calculated and this current length is demonstrated in the window 7. One of such intermediate routes is displayed in the map, Fig. 4. As told, a pause for the time *dT* is made after each route. After all the *P* are over, all the lengths become gathered in a numeric array *allLength* for which the last operation is produced

[*ShortestWay*, *I*] = ***min***(*allLength*)

that returns the shortest way among all possible for this current *CityCollection* (4), and its number *I* within the permutations *P*. For the *I* found, we plot by green the final, i.e. the shortest route that provides the solution. In the window 7 the *ShortestWay* is finally displayed.

5. Research of the time consumption. The famous feature of the TSP problem is the drastic increase of the solution time *T* while the number of cities *M* grows. As an example: the time *T*(10) is, say, equal to 1 minute for *M*=10; the time for *T*=11 will be *T*(11)=11**T*(10), i.e. about 10 minutes. Moreover, the next case for *N*=12 cities will consume the time *T*(12)=12**T*(11), i.e. about 120 minutes, or 2 hours. Next test complexity *M*=13, easy to see, will consume 13*2=26 hours, or about 2 days.

Such behavior of the program, T~M!, should obviously be demonstrated to students. A certain numerical research in MATLAB with our program may be recommended to them by means of surrounding it by commands *tic* and *toc* that run stopwatch timer. Having collected data for several *M*, it is worth to analyze the graph *T*=*T*(*M*). Note that *dT* must be set to 0 in the window 4, Fig. 2, for such research. Restriction of our program was 20 cities.

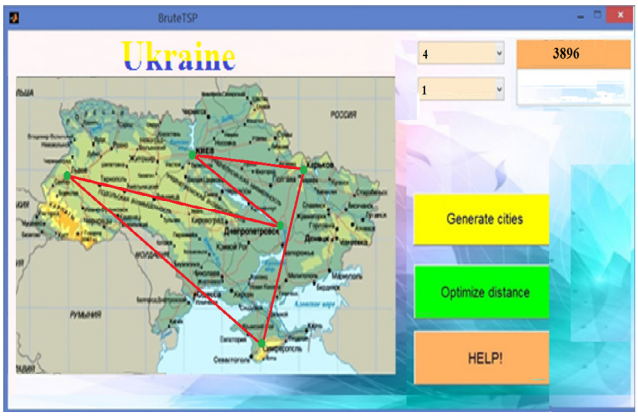

Fig. 4. Program views one of current routes for a task with four cities (beside origin city 0) at a certain time delay 1

That is why this “naive” method was called “brute-force algorithm”, and various efforts were made into invention of more heuristic but realistic algorithms. Several such algorithms were elaborated, that found their recent applications in effective popular automotive GPS navigators [14]. However, the TSP still remains unsolved “in full”, [11-13,15]. The problem’s state of the art has been completely described in popular resources mentioned. It is not surprising that the TSP is also topical for aero navigation as well.

6. Conclusion

Despite the Traveling Salesman Problem (TSP) is one of the most difficult in discrete mathematics, its short synopsis was suggested to include in educational courses for students in aero navigation area. MATLAB was considered as one of most appropriate mathematical workbenches for this. The program *BruteTSP.m* was developed and briefly explained as a simple but modern GUI-program that manages several other programs and algorithms to realize “brute-force method” in the search of shortest route between given number of cities. Ukrainian map and several collections of Ukrainian cities were used to demonstrate the program. Several programs employed belong to key stone algorithms of the Computer Science.

As told, we “honestly” enumerated all the possible routes (the “Brute-force Method”). We find it methodologically correct to research this method first; students will have a comparison base if go on with other methods, more heuristic.

References

[1] Azarskov V.M., Gayev Ye.A. (2014) *Suchasne programuvannya* [*Modern programming*, p. 1. Kyiv: NAU. – 256 p.]. (in Ukrainian)

[2] Gayev E.O., Azarskov V.M. (2016) *Suchasne programuvannya* [*Modern Programming*, p. 2. Kyiv: NAU. – 197 p.]. (in Ukrainian)

[3] Azarskov V.M., Gayev E.A. (2015) Programmirovanie intensifitsiruet ovladenie nauk studentami [Programming intensifies mastering sciences by students]. *Materiali XIY Mizhnarodnoji Konferencii* [Materials XIY International Conference «Industrial hydraulics and pneumatics»], Sumy, pp. 28 -29. (in Russian)

[4] Gayev E.A., Martych. M., Tarak G. (2015) Programmy modelirovaniya sluchainykh yavlenii dlya izucheniya programmirovaniya i matematiki [Programs for modeling random phenomena for learning programming and mathematics]. *Information Technologies in Education*, 2015, № 23, p. 30-42. (http://ite.kspu.edu/webfm_send/829) (in Russian)

[5] Gayev E.A., Rozhok O., Ovcharchin N. (2014) Zvuk ta muzyka v kursi prohramuvannia [Sound and music in the course of programming]. *Software Engineering*, 2014, № 3(19), pp. 41 - 48. (in Ukrainian)

[6] ttps://en.wikipedia.org/wiki/List_of_educational_programming_languages

[7] Discovery-Based Learning. https://en.wikipedia.org/wiki/Discovery_learning

[8] Entdeckendes Lernen. https://de. wikipedia.org/wiki/Entdeckendes_Lernen. (in German)

[9] An algorithmic programming language developed within Russian Buran space project. https://en.wikipedia.org/wiki/DRAKON

[10] Computer Science Curriculum 2008. IEEE Computer Society. http://www.acm.org//education/curricula/ComputerScience2008.pdf

[11] A full and modern overview of the Travelling Salesman Problem (TSP). https://en.wikipedia.org/wiki/Travelling_salesman_problem

[12] Golovnev A.G. (2012) *Priblizhennye algoritmy resheniya perestanovochnykh zadach.. Diss. magistra.* [Approximation algorithms for solving commuting problems]. - Master degree Thesis, SPb.: Academic Educ.-Sci. Nanotechnology Center of RAS. - 25 p. (in Russian)

[13] Mudrov V.I. (1969) Zadacha o kommivoyazhere. [*The problem of the traveling salesman*.]. Moscow: "Znanie", 62 p. (in Russian)

[14] An overview of GPS technique. https://en.wikipedia.org/wiki/GPS_navigation_device

[15] TSP overview in German. https://de.wikipedia.org/wiki/Problem_des_Handlungsreisenden.

Received 02 May 2017

Є.О. Гаєв, В.В. Калмиков
Задача комівояжера в освіті інженерів-програмістів

Мета: Славетну задачу комівояжера (ЗК) пропонується зробити звичайною вправою для студентів інженерних спеціальностей за умов, що вони оволодівають комп'ютерними науками в середовищі легкого програмування, як MATLAB. **Методи:** легке програмування в MATLAB дозволяє такий сучасний педагогічний підхід, як "навчання на власних відкриттях". **Результати:** MATLAB-програма для ЗК, орієнтована на карту України, яка дозволяє графічно ілюструвати процес пошуку оптимального шляху між містами з опцією прискорення або гальмування демонстрації. Очікуємо, що програма буде корисною для вивчення ЗК як однієї з фундаментальних логістичних задач, а також як захоплююча вправа з програмування. Запропоновано, крім того, кілька підпрограм, що належать до ключових алгоритмів курсу програмування. Таке поєднання відповідає сучасному педагогічноиу методу "вчитися шляхом відкриттів". **Обговорення:** Ми роз'яснюємо, як створити візуальну програму для дискретної оптимізації, пропонуємо потрібні підпрограми, включаючи доволі сучасний графічний інтерфейс користувача, пропонуємо її використання для демонстрації катастрофічного росту часу, потрібного для знаходження розв'язку. **Висновки:** Легке програмування, реалізоване в MATLAB, перетворює складні програмні задачі у привабливі, зосереджує студентів на точній постановці задачі, потрібних законах та алгоритмах, реалізуючи таким чином педагогічний підхід "навчання шляхом відкриттів".

Ключові слова: ЗК (задача комівояжера); комбінаторика; логістика; навчання програмуванню; оптимізація; навчання шляхом відкриттів; програмування.

Е.А. Гаев, В.В. Калмыков
Задача коммивояжера в образовании инженеров-программистов

Цель: Знаменитую задачу комивояжера (ЗК) предлагается сделать обычным упражнением для студентов инженерных специальностей при условии, что они овладевают компьютерными науками в среде легкого программирования, как MATLAB. **Методы:** легкое программирование в MATLAB позволяет осуществить такой современный педагогический подход, как "обучение на собственных открытиях". **Результаты:** MATLAB-программа для ЗК, ориентированная на карту Украины, которая позволяет графически иллюстрировать процесс поиска оптимального пути между городами с опцией ускорения или задержки демонстрации. Ожидаем, что программа будет полезной для изучения ЗК как одной из фундаментальных логистических задач, а также как увлекательное упражнение по программированию. Предлагается также несколько подпрограмм, принадлежащих к ключевым алгоритмам программирования. Такое соединение отвечает современному педагогическому методу "учиться путем открытий". **Обсуждения:** Мы разъясняем, как создать такую визуальную программу с довольно современным графическим интерфейсом пользователя, предлагаем ее использование для демонстрации катастрофического роста времени, необходимого для нахождения решения. **Выводы:** Легкое программирование, реализованное в MATLAB, преобразует сложные задачи в увлекательные, фокусирует студентов на точной постановке задачи, необходимых законах и алгоритмах, реализуя таким образом педагогический подход "обучения путем открытий".

Ключевые слова: ЗК (задача коммивояжера); комбинаторика; логистика; обучение путем открытий; обучение программированию; оптимизация; программирование.

Yevgeny Gayev. Doctor of Engineering. Professor.
Professor on Aircraft Control Systems Department, National Aviation University, Kyiv, Ukraine.
Education: Kharkiv State University, Kharkiv, Ukraine (1971).

Research area: fluid mechanics and thermal physics, boundary layer theory, mathematics, computer science, history of science.
Publications: 160.
E-mail: Ye_Gayev@i.ua

Vadim Kalmikov (1999). Student.
Aircraft Control Systems Department, National Aviation University, Kyiv, Ukraine.
Research area: computer science, aerodynamics.
Publications: 3.
E-mail: kvvad@ukr.net