Optimal solution of the Guarini puzzle extension using tripartite graphs

A E Iordan
Politehnica University of Timisoara, Department of Electrical Engineering and Industrial Informatics, 5 Revolution Street, Hunedoara, 331128, Romania
E-mail: anca.iordan@fih.upt.ro

Abstract. In this paper there will be presented a software implemented in the C# programming language using .Net Frameworks platform which allows an optimal solution of a puzzle obtained through Guarini puzzle extension. The generalization of Guarini puzzle consists in considering $2n$ knights placed on the chessboard of $4n$ dimension. Half of these knights are white and are positioned on the first line of the chessboard, and the other half are black and are positioned on the last line of the chessboard. The aim of this puzzle is to move the knights through a minimum number of moves so on the first line we have all the black knights, and on the last line we have all the white knights. This puzzle belongs to a category of problems which can be efficiently resolved by use of tripartite graph. Numbering the chessboard squares these will represent the vertices of the graph, and the possible moves of the knights will represent the edges of the graph. Modelling this problem associated with this puzzle, it can be noticed that the corresponding vertices of the squares from the first and last line form the first vertices subset from the tripartite graph, the ones corresponding to the second line of squares form the second vertices subset from the tripartite graph, and the ones corresponding to the third line of squares form the third vertices subset from the tripartite graph.

1. Guarini puzzle generalization
The generalization of Guarini puzzle [1], [2] consists in considering $2n$ knights placed on the chessboard of $4n$ dimension: the $n$ white knights are at the first line, and the $n$ black knights are at the last line. The scope of this puzzle is to exchange the knights to obtain the position represented on the right side of figure 1 (particular case $n=3$) in the minimum number of knight moves, not permitting more than one knight on a square at any time.

This puzzle belongs to a class of problems which can be efficiently resolved by use of tripartite graph [3]. Numbering the chessboard squares these will represent the vertices of the graph, and the possible moves of the knights will represent the edges of the graph. Modelling this problem associated with this puzzle, it can be noticed that the corresponding vertices of the squares from the first and last line form the first vertices subset from the tripartite graph, the ones corresponding to the second line of squares form the second vertices subset from the tripartite graph, and the ones corresponding to the third line of squares form the third vertices subset from the tripartite graph.

2. Tripartite graphs
In graph theory a multipartite graph is a graph whose vertices can be divided into $n$ different independent sets [4]. In other words, it is a graph that can be colored with $n$ colors, so that no two endpoints of an edge have the same color.
Figure 1. Start state and goal state for the six knights puzzle

When $n=2$ these are the bipartite graphs [5] (Figure 2), and when $n=3$ they are called the tripartite graphs (Figure 3) [6]. A complete multipartite graph [7] is a multipartite graph in which there is an edge between every pair of vertices from different independent sets. These graphs are described by notation with a capital letter $K$ subscripted by a sequence of the sizes of each set in the partition. For example, $K_{2,2,2}$ is the complete tripartite graph of a regular octahedron (Figure 4), which can be partitioned into three independent sets each consisting of two opposite vertices.

Figure 2. Example of a bipartite graph

Figure 3. Example of a tripartite graph
3. The analysis of the software

The scope of this paper is to present a software which permits an optimal solution of the puzzle presented in the previous paragraph. The mathematical modelling of this puzzle with the aim of an optimal resolving can be used to teach tripartite graph notion [8].

From the perspective of unified modelling language [9], the analysis of didactical game application includes the representation of the use-case diagram [10]. The use-case diagram offers simplified and graphical representation of what the didactical software must really do. Use-case diagram (Figure 5) is based upon functionality and thus will focus on the “what” offers the didactical software and not how will be realized.

![Complete tripartite graph K₂,₂,₂](image)

**Figure 4.** Complete tripartite graph K₂,₂,₂

![Use-case diagram](image)

**Figure 5.** Use-case diagram
The use-case diagram corresponding to this didactical software includes:
- One actor - the user who is external entity with that the software interacts;
- Eight use-cases which describe the performance of the software;
- Association relationships between player and use-cases, and dependency relationships between use-cases.

4. The design of the software
Conceptual modelling allows identifying the very important elements for the didactical software application. Class diagram [11] is represented (Figure 6) in order to be observed the connection mode between the classes and the interfaces that are used and also the composition and aggregate relationships between instances.

After identifying the specific elements of the didactical software, they were been implemented 3 interfaces and 9 classes grouped into three packages.

![Class diagram](image-url)
Package “GameState” brings together concepts that correspond to the algorithm corresponding to the tripartite graph, consisting of 2 classes and 2 interfaces. For memorizing a state configuration, there was implemented “StateGame” class which implements “IStateGame” interface.

The algorithm corresponding to tripartite graph is implemented by using “Algorithm” class which implements “IAlgorithm” interface. An instance of this class is composed by one instance of “StateGame” class and one instance of “TripartiteGraph”, according to composition relationship which exists in diagram.

Package “GraphicalUserInterface” brings together concepts that correspond to the graphical user interface corresponding to the didactical software, consisting of 3 classes. For designing the graphical interface of the software, it was implemented the „MainInterface” class.

An instance of this class is composed by one instance of “TripartiteClassRepresentation” class and one instance of “StateGameRepresentation” class, according to composition relationship which exists in diagram. The “StateGameRepresentation” class was implemented for show the current state of the puzzle, and the “TripartiteGraphRepresentation” class was implemented to present the tripartite graph associated to the current state of the puzzle.

Package “Graph” (Figure 7) brings together concepts that correspond to the graph used to solve the puzzle, consisting of 4 classes and one interface. The concept of a vertex from a graph is implemented by using “Vertex” class which implements “GraphicalRepresentation” interface. The concept of an edge from a graph is implemented by using “Edge” class which implements “GraphicalRepresentation” interface. An instance of this class is composed by one instance of “Vertex” class.

The concept of graph is implemented by using “Graph” class which implements “GraphicalRepresentation” interface. An instance of this class is composed by several instances of “Vertex” class and several instances of “Edge” class, according to composition relationship which exists in package diagram. The concept of tripartite graph is implemented by using “TripartiteGraph” class which inherits from “Graph” class.

In the design phase, it also performs statechart diagrams that specify the dynamic behavior of class instances. Figure 8 shows the statechart diagram associated with an instance of the “Algorithm” class.

5. Graphical user interface
The software is accomplished using the C# object-oriented programming language [12], [13] on the .Net Framework platform [14], [15]. Given that specified requisites in uses cases diagram (Figure 5) it was designed graphical user interface.

After the level of difficulty was selected, it is generated the tripartite graph associated to the start state (Figure 9), and the player can begin the game. The application permits moves of any knight between two adjacent vertices, such that any vertex of the tripartite graph does not contain more than one knight at any time.

After each move, the current state of the puzzle is updated (Figure 10) and it is checked whether the goal state has been reached.

When the goal state is reached (Figure 11), it is checked whether the solution obtained is optimal.

6. Conclusions
Because the graph theory is a difficult field of computer science, the computer games can be of helpful in the teaching/learning process of manifold aspects regrading to graphs. The utilization of didactical software signifies an efficient way of stimulation and development of the student motivation.

The didactical software presented in this paper contributes to the qualitative assimilation of the knowledge and to the competences development related to the tripartite graphs, having great educational opportunities, because it shows the most accessible and the most interesting type of activity for students.
Figure 7. Package diagram
Figure 8. Statechart diagram for an instance of the “Algorithm” class.

Figure 9. Graphical user interface corresponding to the start state of the 6 knights puzzle.

Figure 10. Graphical user interface corresponding to an intermediate state.
Figure 11. Graphical user interface corresponding to the goal state of the 6 knights puzzle

References
[1] Levitin A and Levitin M 2011 Algorithmic Puzzles, Oxford University Press, New York, USA
[2] Grabarchuk S 2005 The New Puzzle Classics: Ingenious Twists on Timeless Favorites, Sterling Publishing Co Inc., pp 204-206
[3] Rajasekaran G and Sampathkumar R 2015 Optimal Orientations of Some Complete Tripartite Graphs, Mathematics Subject Classification 29(8) 1681-1687
[4] Even S and Even G 2012 Graph Algorithms, Cambridge University Press, New York, USA
[5] Muscalagiu I, Vidal J and Cretu V 2007 Experimental Analysis of the effects of agent synchronization in Asynchronous search algorithms, Kes International Symposium on Agent and Multi-Agent, Wroclaw, Poland, May 31- Jun 1, pp 619-636
[6] Diestel R 2017 Graph Theory, Springer Verlag, New York, USA
[7] Rob R, Panoiu C and Panoiu M 2015 Voltage and Current Signals Processing in Functioning of Nonlinear Loads, Soft Computing Applications 2 1171-1184
[8] El-Zohny H and El-Morsy H 2012 Shortest Path Algorithm for some Graphs before and after Folding, International Journal of Pure and Applied Mathematics 75(2) 141-148
[9] Bruegge B and Dutoit A 2013 Object Oriented Software Engineering Using UML, Prentice-Hall, Inc. Upper Saddle River, NJ, USA
[10] Dennis A, Wixom B H and Tegarden D 2012 Systems Analysis and Design with UML, John Wiley & Sons Ltd, Hoboken, NJ, USA
[11] Rumpe B 2016 Modeling with UML, Springer International Publishing AG, Switzerland
[12] Doyle B 2013 C# Programming From Problem Analysis to Program Design, Cengage Learning, Australia
[13] Purdum J 2012 Beginning Object Oriented Programming with C#, John Wiley & Sons, Inc., Indianapolis, Indiana, USA
[14] Gittleman A 2011 Computing with C# and the .Net Framework, Jones and Bartlett Publishers, Burlington, MA 01803, USA
[15] Thuan T and Hoang L 2015 .Net Framework Essentials, O'Reilly & Associates, Inc. Sebastopol, CA, USA