

Introduction to Nondeterminism

Gabriela Lovászová, *glovasz@ukf.sk*

Dept of Informatics, Constantine the Philosopher University in Nitra, Slovakia

Viera Palmárová, *vpalmarova@ukf.sk*

Dept of Informatics, Constantine the Philosopher University in Nitra, Slovakia

Abstract

The term of nondeterminism discussed in this paper is usually not included in the introductory programming courses. Secondary school students are familiar with the idea that programs have to be deterministic because they are processed by computers automatically. Later, when they come to the university, it is difficult for them to understand the concept of nondeterminism. In our opinion, the gentle introduction to nondeterminism may be done even at the secondary school level.

A good motivation for introducing the nondeterminism and nondeterministic algorithms might be some logic-based puzzle games. We mean those puzzles, which are really difficult to solve by deterministic algorithms. One needs to use logical reasoning and several strategies and their combination to make right decisions when constructing a solution of such puzzles.

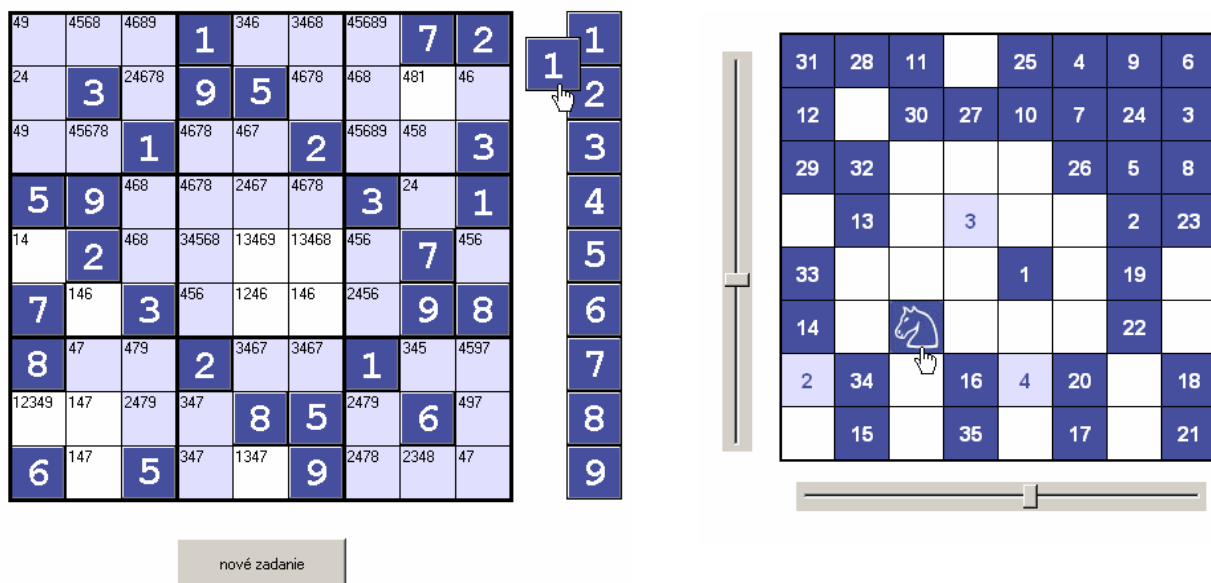


Figure 1. Screenshots of the Imagine Logo projects

In this paper, two Imagine Logo projects are presented. They provide interactive environments for solving two well-known logical puzzles – Sudoku and Knight's Tour. However, the main goal of these projects is not to find the solution but to assist a player in making good decisions when executing a nondeterministic algorithm. We want students to construct their understanding of the nondeterminism concept on the basis of their experience.

Keywords:

nondeterminism; computational complexity; heuristic algorithms; Imagine Logo; Sudoku; Knight's Tour problem

1 Introduction

The theory of computation and complexity theory are the fundamental parts of theoretical computer science. The first one deals with the question which problems are algorithmically solvable and which problems cannot be solved by any algorithm. The complexity theory provides methods and tools of quantitative measures enabling us to compare the efficiency of algorithms and to classify computing problems according to their computational hardness.

Algorithms can be evaluated and compared by a variety of criteria. Most often we are interested in the time or the space required to solve some problem by given algorithm. Both the time and the space differ from input to input, that means the algorithm's running time and memory space are functions of the input. The time complexity of an algorithm is measured by the number of elementary operations executed by the algorithm processing the given input. The space complexity of an algorithm is the number of elementary memory cells used in the computing process on some input. The computational complexity of the problem is the complexity of the best (optimal) algorithm solving the given problem.

We can classify problems according to their computational complexity into tractable problems admitting polynomial-time solution and intractable (computationally difficult) problems for which no polynomial algorithm is known. As computers have a finite memory and we have a finite time for waiting for a result, the complexity function of the algorithm bounds the size of inputs which can be processed. For that reason intractable problems are solvable in practical use only for some small inputs. Nevertheless there may be a large practical interest to have a program solving the hard intractable problems.

Some of these difficult problems become tractable using nondeterminism. We say that such algorithm is nondeterministic, in that some computing steps are optional and the algorithm guesses which of the available option is better. Consequently, it generates various computations for the given input. We can represent all possible computations of a nondeterministic algorithm on a given input as a tree. A deterministic computation, in contrast, is a linear sequence of computing steps.

How can nondeterminism help us to solve difficult problems effectively? Many intractable problems admit short polynomial-time certificates that means we can verify whether the solution is correct in real time. Such problems may be solved by polynomial nondeterministic algorithm, which can guess the solution and then check in reasonable time whether this guess was correct. Unfortunately, a deterministic simulation of the work of nondeterministic algorithm is usually a search for a solution in the computation tree and it usually causes an exponential increase of time complexity. In spite of this we can use the following approaches to obtain a solution in reasonable time (Calude and Hromkovic, 1997):

- Deterministic algorithm. If the inputs are in the range where time-complexity is not too large, then deterministic algorithm (despite of its exponential complexity in general) is a useful solution for hard problem.
- Problem restrictions. In several cases, we do not need to solve the given problem in generality and some restrictions on original formulation decrease the computational difficulty of the problem.
- Approximate algorithms. If we have a hard optimization problem, we can solve it easier searching for a solution which is not necessarily the optimum, but not too far from the optimum.
- Probabilistic algorithms. They are based on nondeterministic ones. Each nondeterministic step is interpreted as a random decision. Instead of surely correct outputs produced by deterministic algorithms probabilistic one computes output whose probability to be correct is large enough.
- Heuristics. They are similar to probabilistic ones; they make decisions during nondeterministic computation. The difference is that it is unable to prove that heuristic

algorithms provide correct solutions despite of the fact that they produce good outputs in practice.

2 Term of nondeterminism in secondary education

Algorithmics and programming are the fundamental and enduring concepts of computer science. They provide long-term knowledge in a rapidly changing discipline, in contrast of short-term computer-specific and application-specific skills. Understanding of these fundamental concepts is essential to effective use of computers and information technology. They also develop and improve students' general intellectual capabilities such as problem solving, logical reasoning and expressing ideas in a formal way. These are the reasons why fundamentals of algorithmics and programming are also a part of general education in secondary schools in Slovakia.

The term of nondeterminism discussed in this paper is usually not included in the introductory programming courses. Secondary school students are familiar with the idea that programs have to be deterministic because they are processed by computers automatically. Later, when they come to the university, it is difficult for them to understand the concept of nondeterminism. In our opinion, the gentle introduction to nondeterminism may be done even at the secondary school level.

A good motivation for introducing the nondeterminism and nondeterministic algorithms might be some logic-based puzzle games. We mean those puzzles, which are really difficult to solve by deterministic algorithms. One needs to use logical reasoning and several strategies and their combination to make right decisions when constructing a solution of such puzzles. Therefore, we suggest the puzzles as a suitable tool for showing the difficulty of intractable problems and the above-mentioned approaches to solving hard problems.

3 Imagine Logo projects

We have prepared two projects which provide interactive environments for solving two well-known logical puzzles. However, the main goal of these projects is not to find the solution but to assist a player in making good decisions when executing a nondeterministic algorithm. We want students to construct their understanding of the nondeterminism concept on the basis of their experience.

3.1 The Sudoku Project

Sudoku is a very popular number placement puzzle. The rules are quite simple: Fill a 9x9 grid so that every row, every column, and every of the nine 3x3 boxes contain each digit from 1 to 9 only once. There are usually some numbers already provided in the grid which represents an initial state and cannot be changed (Figure 2).

			1				7	2
	3		9	5				
		1			2			3
5	9					3		1
	2						7	
7		3					9	8
8			2			1		
				8	5		6	
6		5			9			

Figure 2. Sudoku puzzle

The Sudoku project implements two strategies that decrease the number of available options when looking for a cell where some digit can be placed. Let us describe the strategies briefly:

Strategy 1: Determine possible numbers for every single cell in the grid. If some cell contains one and only number, then that is the right solution for the cell. Figure 3 illustrates the output of scanning the grid for the puzzle depicted on Figure 2. Unfortunately, there is no cell with the only candidate to write in.

4,9	4,5,6, 8	4,6,8, 9		3,4,6	3,4,6, 8		4,5,8	
2,4		2,4,6, 7,8			4,6,7, 8	4,6,8	1,4,8	4,6
4,9	4,5,6, 7,8		4,6,7, 8	4,6,7		4,5,6, 8,9	4,5,8	
		4,6,8	4,6,7, 8	2,4,6, 7	4,6,7, 8		2,4	
1,4		4,6,8	3,4,5, 6,8	1,3,4, 6,9	1,3,4, 6,8	4,5,6		4,5,6
	1,4,6		4,5,6	1,2,4, 6	1,4,6	2,4,5, 6		
	4,7	4,7,9		3,4,6, 7	3,4,6, 7		3,4,5	4,5,7, 9
1,2,3, 4,9	1,4,7	2,4,7, 9	3,4,7			2,4,9		4,7,9
	1,4,7		3,4,7	1,3,4, 7		2,4,8	2,3,4, 8	4,7

Figure 3. Strategy 1

Strategy 2: Determine possible cells for a given number. If some cell is the only possible cell in a row, a column or a box, then the given number can be placed in it. Figure 4 illustrates the output of scanning the grid for cells, which the number 1 can contain.

			1				7	2
	3		9	5				
		1			2			3
5	9					3		1
	2						7	
7		3					9	8
8			2			1		
				8	5		6	
6		5			9			

Figure 4. Strategy 2 applied for the number 1

The blue-coloured cells cannot contain the number 1. Notice the cell in the second row and eighth column. It is the only not coloured cell in its row, column and box, what makes this cell an acceptable place for the number 1.

It depends on difficulty level of the puzzle whether we can eliminate candidate digits for cells to get just one choice and find the solution in a deterministic way or whether we do need to do some nondeterministic decisions and then use the trial-and-error method as well.

The generalization of Sudoku puzzle to larger grids ($n^2 \times n^2$ board of $n \times n$ blocks) is known to be NP-complete problem (Yato, 2003).

3.2 The Knight's Tour Project

The second puzzle is concerned with a knight chess piece. The knight starts on an arbitrary chessboard's square and proceeds to move according to the rules of chess (*a chess knight moves along an L-shaped path: two squares horizontally and one square vertically, or two squares vertically and one square horizontally*) in order to visit each one of the remaining squares exactly once. The question is, whether it is possible for the knight to make such a tour around the board.

The very first idea is to test all possible ways that the knight could follow while touring. When the knight lands on a square from which it cannot get further without getting to an already visited square, it takes the last move back and tries another direction. Theoretically, if there is a solution, it will be certainly discovered. However, due to exponential complexity of the backtracking algorithm, this strategy does not help. For the classical 8x8 chessboard, it would take too long, to get a result.

In our project, an old but efficient heuristic method known as the Warnsdorff's rule is implemented. To build the solution quickly, we have to avoid creating dead ends. For that reason the possible squares to be chosen next are examined before every move. During the knight's tour, the number of free entrances to all squares of the board is gradually used up. Warnsdorff's rule serves to direct the knight to the squares with the least numbers of free entrances left - before these squares have turned into dead ends (Törnberg, 1999).

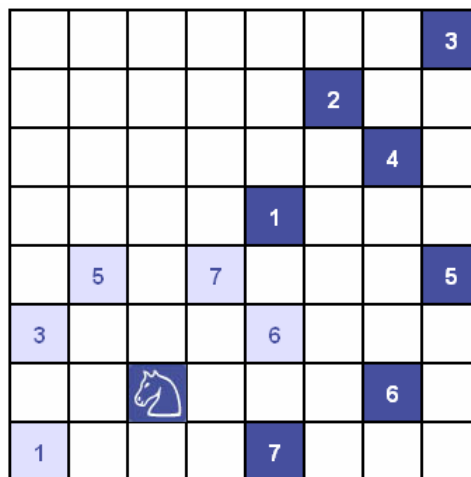


Figure 5. The knight is waiting until we make the next move

Take a look at the situation depicted in Figure 5. When the knight is standing on a square, all next legal moves are highlighted. For each one of the light blue squares, the number of free entrances is counted and displayed. Now, the knight should be moved to the square with the lowest number. Dark blue squares are the ones already visited. Each dark blue square is labelled with the proper ordinal number.

The Warnsdorff's rule gives solutions, but not all possible solutions. One can make moves opposing the rule and yet get a complete tour. In many cases there are equal choices and sometimes one might spare a square with the lowest number (in order to have it as the end square of a re-entrant tour). There is a certain limited freedom of choice, but squares with one entrance are due to be visited at once - otherwise they will turn into dead ends. One such square can serve as ending square for the whole tour, two means trouble.

It is possible to refine the Warnsdorff's rule so that it fails less often: if two or more possible next squares are equally good, then the one farthest away from the centre of the board should be chosen.

In graph theory, the knight's tour problem is an instance of the more general Hamiltonian path problem. And if we demand the knight's tour to be closed, we get an instance of the Hamiltonian cycle problem. Both of the graph problems are NP-complete. However, unlike the general Hamiltonian path problem, the knight's tour problem can be solved in linear time (Conrad et al., 1994).

4 Inside the projects

In both described projects, a new base class named *Card* is defined. This *Card* class extends the Imagine Logo built-in *Turtle* class. The auto dragging attribute of every *Card* object is on, its pen is up. After dropping the card inside the predefined area, it is automatically aligned to grid. The Sudoku puzzle and the Knight's Tour puzzle are played on similar square boards but differ a lot as for their rules. So there are another two classes derived from the general *Card* class. In both cases, a special functionality is added. The *SudokuCard* object can be put into a cell only if it is a card with proper number. The *ChessCard* object can be placed on a square only as a result of a legal chess knight move. The general *Card* class could also be used as the base class in other games played with cards on some kind of grid.

5 Conclusion

The teacher's role is to provide environment in which learners solve problems with the aim of learning and understanding something new. Our methodology focuses on the fact that solving process of the discussed puzzles is not deterministic. Learners gain valuable experience in nondeterminism whilst solving the puzzles. This is what they should learn:

- algorithm is nondeterministic when it generates computations with some optional steps,
- nondeterministic algorithm generates various computations for a given input, which may be represented as a tree,
- deterministic time corresponds to the complexity of the search for a solution while nondeterministic time corresponds to the complexity of verifying whether a given solution is correct.

The Imagine Logo development environment offers everything needed to implement an interactive and visually attractive educational application (e. g. object orientation, event-driven programming, multimedia support). Therefore, we used it for our projects as well (Hrušecká and Kalaš, 2006).

References

- Calude, C. and Hromkovic, J. (1997) *Complexity: A Language-Theoretic Point of View*. In Handbook of Formal Languages, G. Rozenberg and A. Salomaa (eds), Volume 2. Springer, pp.1-54.
- Conrad A., Hindrichs T., Morsy H. and Wegener I. (1994) *Solution of the Knight's Hamiltonian Path Problem on Chessboards*. Discrete Applied Math, volume 50, no.2, pp.125-134.
- Hrušecká, A. and Kalaš, I. (2006) *Programovanie v prostredí Imagine*. MPC v Bratislave, Bratislava.
- Törnberg, G. (1999) *Warnsdorff's rule*. <http://web.telia.com/~u85905224/knight/eWarnsd.htm>
- Yato, T. (2003) *Complexity and Completeness of Finding Another Solution and its Application to Puzzles*. Master thesis, University of Tokyo, Tokyo.