Sign In | Sign Out | ASME.org Home

eBooks/International Conferen...Using Control Network ...

A+ A- A=

Home | Search | Journals | Proceedings | eBooks | Article Pack |

EBOOKS

Year
Subject
Title

DIGITAL LIBRARY

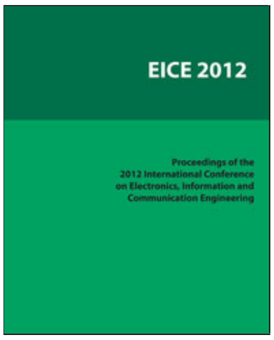
Home
Search
Journals
Proceedings
eBooks
Subscription Information
Feedback
ASME DL Tour
Help

SCITATION

Scitation FAQ
Scitation Home
Scitation Search
Search SPIN
MyScitation

Paper 14, Using Control Network Programming in Teaching Randomization

In Advanced Technology in Electronics and Computersfrom:International Conference on Electronics, Information and Communication Engineering (EICE 2012)
Author(s)/Editor(s): **Garry Lee**


EICE 2012
Proceedings of the 2012 International Conference on Electronics, Information and Communication Engineering

Published: **2012**
Author(s)/Editor(s): **Kostadin Kratchanov, Emilia Golemanova, Tzanko Golemanov, Burcu Külahçıoğlu**
Chapter DOI:
http://dx.doi.org/10.1115/1.859971.paper
Chapter Page Count: **6 pages**

Excerpt

A series of two reports are presented to this conference. The aim of the series is to demonstrate that Control Network Programming, respectively Spider, can be used as an excellent environment for teaching and learning both non-determinism and randomization. More specifically, the emphasis is on CNP implemented models and algorithms typically studied in courses on the Computation theory and Artificial intelligence for students in computing programs. While the focus of first of the reports was using CNP in teaching and learning the concept of non-determinism, the current presentation addresses probabilistic computation models and randomized algorithms.

http://dx.doi.org/10.1115/1.859971.paper14

BOOK DATA

Print ISBN:

9780791859971

Publisher:

ASME Press

KEYWORDS

Control network programming, teaching computing, randomness, randomized models, randomized algorithms, probabilistic models, probabilistic computation, computation models, algorithms, computation theory, artificial intelligence, hill-climbing, search algorithms

Proceedings of 2012 International Conference on Electronics, Information and Communication Engineering
EICE 2012
March 1-2, 2012, Macau

USING CONTROL NETWORK PROGRAMMING IN TEACHING RANDOMIZATION

Kostadin Kratchanov / Department of Software Engineering, Yaşar University, Universite Cad. No.35-37, Bornova/Izmir, 35100 Turkey

Emilia Golemanova / Department of Computing, Ruse University, 8 Studentska Street, Ruse,

Tzanko Golemanov / Department of Computing, Ruse University, 8 Studentska Street, Ruse,

Burcu Külahçıoğlu / Department of Software Engineering, Yaşar University, Universite Cad. No.35-37, Bornova/Izmir, 35100 Turkey

ABSTRACT

A series of two reports are presented to this conference. The aim of the series is to demonstrate that Control Network Programming, respectively Spider, can be used as an excellent environment for teaching and learning both nondeterminism and randomization. More specifically, the emphasis is on CNP implemented models and algorithms typically studied in courses on the Computation theory and Artificial intelligence for students in computing programs. While the focus of first of the reports was using CNP in teaching and learning the concept of nondeterminism, the current presentation addresses probabilistic computation models and randomized algorithms.

Keywords: Control network programming, teaching computing, randomness, randomized models, randomized algorithms, probabilistic models, probabilistic computation, computation models, algorithms, computation theory, artificial intelligence, hill-climbing, search algorithms.

INTRODUCTION

This series of two reports presents suggestions on how Control Network Programming (CNP) can be effectively used in teaching and learning the concepts of nondeterminism and randomization in computing curricula. The first report [1] contained an introduction to the concept of nondeterminism, an overview of software tools used in teaching and learning nondeterministic models, a brief summary of Control Network Programming (CNP) as a programming style combining procedural and declarative programming paradigms which is especially well suited for solving problems that can be easily visualized as set of graphs and/or include nondeterminism. The aim of the first report was to demonstrate how CNP can be conveniently and effectively used in creating and studying nondeterministic models and algorithms studied in university courses on the Theory of computation, Artificial Intelligence, Algorithm analysis and design, etc.

This current presentation is the second report of the series. It focuses on the usage of CNP in teaching and learning the concept of randomization, more specifically, randomized (also referred to as probabilistic) computation models and algorithms.

Nowadays randomized models and algorithms form a standard, core material in computing curricula. It has been even suggested that randomization is studied first and used as a basis for introducing nondeterminism as its special case [2] (although we do not share this opinion). At the same time understanding and apprehending the concept of probabilistic computation poses a substantial challenge to the students. Our presentation below shows how CNP can be conveniently and effectively used as a teaching and learning tool in support of the learner.

The report follows the terminology introduced in [1].

1 RANDOMNESS

The notion of randomness is one of the most fundamental and most discussed terms in science [3]. In a randomized computation [4, 5], similarly to nondeterministic ones, there are certain points where a choice is to be made of the way to proceed. For simplicity, we may assume that there are exactly two possible moves - then, the probability of each choice is ½. A randomized computation differs from a nondeterministic one in the fact that a single random choice must be made at each choice point, a single computation path is realized, choices cannot be revoked and therefore no other computation path is considered. Random guesses can be implemented reasonably efficiently in practice, in contrast to the exhaustive search assumed in nondeterministic algorithms.

Note [3, 6] that there is an important difference between nondeterministic and probabilistic machine models. Nondeterministic models [1] are simply useful abstract theoretical notions and are not meant to be actually implemented. They can be intuitively seen as having the ability to make nondeterministic guesses such that computation time is minimized. Probabilistic machine models, on the other hand, are meant to be implemented. They can be seen as making actual random choices; consequently, each possible path of computation corresponds to a sequence of such mutually independent random choices and has a probability associated with it.

Randomized algorithms may (but not necessarily) be more efficient or simpler than their deterministic counterparts. It is known that for several important problems there is an efficient polynomial-time randomized algorithm while at the same time no tractable deterministic algorithm has been found. Therefore, randomized algorithms have become a standard tool in almost all areas of computer science.

2 (LACK OF) SOFTWARE TOOLS FOR TEACHING RANDOMIZATION

Unlike educational software tools for nondeterministic models [1], we are not aware of general-purpose tools to be used for teaching randomization and probabilistic models.

A few tools (already discussed in [1]) can be mentioned that are applicable in certain very special cases. Probabilistic automata can be studied using UPPAAL Pro [7], a probabilistic extension to UPPAAL. Matlab has certain random number generator functions; they can be used to generate some randomized models.

3 CNP IN TEACHING RANDOMIZATION

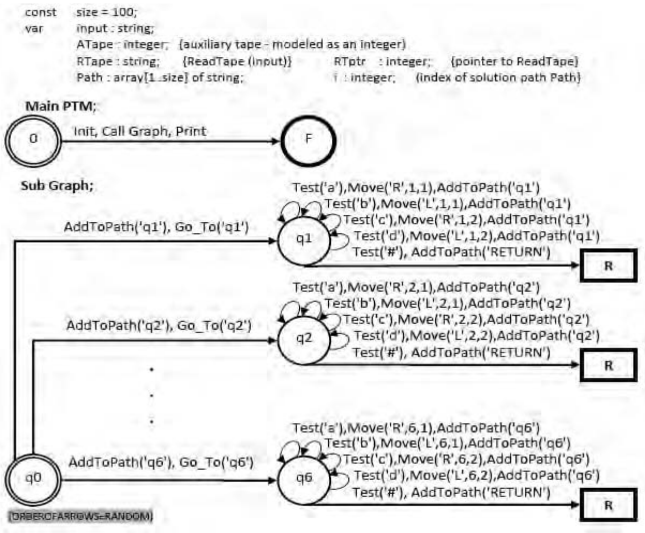
As shown in [1], CNP is an excellent tool for teaching and learning nondeterminism. It can be also successfully used in teaching randomization. One straightforward possibility is to use a primitive which generates random numbers. We could call such implementations procedural solutions [1]. There is also a second, non-standard possibility – the design of non-procedural [1] implementations with the usage of the system options ORDEROFARROWS or RANGEORDER with value RANDOM [8, 9]. The approach for non-procedural implementation of randomness is similar to the one applied for non-procedural implementation of local search strategies [10-12].

IMPLEMENTING RANDOMIZED MODELS IN COMPUTATION THEORY AND ALGORITHM DESIGN

To demonstrate our approach, we describe below two different implementations of a randomized (also called probabilistic) Turing machine. The first solution is non-procedural and unique for the CNP approach, while the second one is procedural and mimics a solution that would normally be used in an imperative programming language.

The probabilistic Turing machine that we program in WinSpider is based on an idea from [13]. It accepted words in which the number of a's is equal to the number of b's, and the number of c's is equal to the number of d's. In the notation from [13] this means that k = 2. This is a one-sided error algorithm; we implement a solution in which r = 3k, consequently, the algorithm may give a wrong positive answer with probability not more than 1/3.

The nonprocedural solution is shown in Fig. 1. This probabilistic TM has an input (read) tape, RTape and an auxiliary tape, ATape. The latter is actually modeled by an integer (in this particular problem that is enough – the integer correspond to the position of the head over the auxiliary tape). Primitives AddToPath(node) and Test(ch) are the same as the ones used in the earlier nondeterministic TM example. Primitive Go_To(node) displays the node (branch) which is chosen randomly from state q0 as result of the setting [ORDEROFARROWS=RANDOM]. Primitive Move(direction,base,exponent) changes the value of ATape in a manner that corresponds to the algorithm from [13]: increases the value of ATape by baseexponent if direction = 'R', or decreases the value by the same amount if direction = 'L'.



```
const   size = 100;
var     input : string;
        ATape : integer;  {auxiliary tape - modeled as an integer}
        RTape : string;      {ReadTape (input)}        RTptr   : integer;    {pointer to ReadTape}
        Path : array[1..size] of string;               i  : integer;     {index of solution path Path}
```

Main PTM;
0
Init, Call Graph, Print
F
Sub Graph;
AddToPath('q1'), Go_To('q1')
q1
Test('a'),Move('R',1,1),AddToPath('q1')
Test('b'),Move('L',1,1),AddToPath('q1')
Test('c'),Move('R',1,2),AddToPath('q1')
Test('d'),Move('L',1,2),AddToPath('q1')
Test('#'), AddToPath('RETURN')
R
AddToPath('q2'), Go_To('q2')
q2
Test('a'),Move('R',2,1),AddToPath('q2')
Test('b'),Move('L',2,1),AddToPath('q2')
Test('c'),Move('R',2,2),AddToPath('q2')
Test('d'),Move('L',2,2),AddToPath('q2')
Test('#'), AddToPath('RETURN')
R
AddToPath('q6'), Go_To('q6')
q0
q6
Test('a'),Move('R',6,1),AddToPath('q6')
Test('b'),Move('L',6,1),AddToPath('q6')
Test('c'),Move('R',6,2),AddToPath('q6')
Test('d'),Move('L',6,2),AddToPath('q6')
Test('#'), AddToPath('RETURN')
R
[ORDEROFARROWS=RANDOM]

Figure 1 Probabilistic Turing machine – nonprocedural solution

The procedural solution is illustrated in Fig. 2. Global definitions and the main subnet are the same as in Fig. 1 and are not shown. The set of primitives used is also the same. The system option ORDEROFARROWS is not used here. Instead, a primitive GetRandom(r) is utilized which generates randomly the value of an integer variable r. This variable is then used as the base of exponentiation in the primitive Move.

The probabilistic machine from our examples implements a Monte Carlo algorithm – it will always correctly recognize a legal input string, but may (with probability <1/6) incorrectly recognize an illegal word. For example, on input accbbb the PTM will incorrectly give a positive result if, in the first version of the PTM state q1 is randomly chosen, or the random value of r is 1 in the second version.

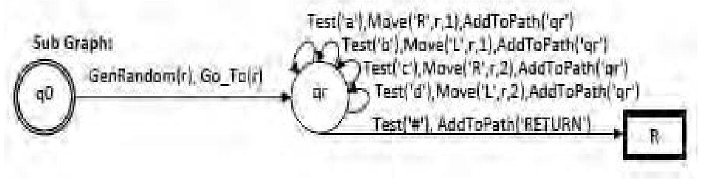

Sub Graph:
q0
GenRandom(r), Go_To(r)
qr
Test('a'),Move('R',r,1),AddToPath('qr')
Test('b'),Move('L',r,1),AddToPath('qr')
Test('c'),Move('R',r,2),AddToPath('qr')
Test('d'),Move('L',r,2),AddToPath('qr')
Test('#'), AddToPath('RETURN')
R

Figure 2 Probabilistic Turing machine - procedural solution

TEACHING RANDOMIZATION IN ARTIFICIAL INTELLIGENCE – RANDOMIZED HILL-CLIMBING SEARCH STRATEGIES

It is well known that the local greedy algorithms (such as hill-climbing) can get stuck in a local optimum. Therefore, various variants have been invented expanding the search space in an effort to avoid becoming trapped in a local optimum [14,15]. This is achieved by incorporating certain form of randomness in the choice of the successor of a state, which in the CNP case means in the choice of the outgoing arrow to follow. Such modifications are stochastic hill-climbing (a next uphill move is chosen randomly), first-choice hill-climbing (successors are generated stochastically and the first of them which is better than the current state is chosen), simulated annealing (a random move is picked; if it is "better" the move is accepted, otherwise the algorithm accepts the move with some probability depending on the steepness of the descent and the "temperature" – bad moves are more likely to be allowed at the start when the temperature is high).

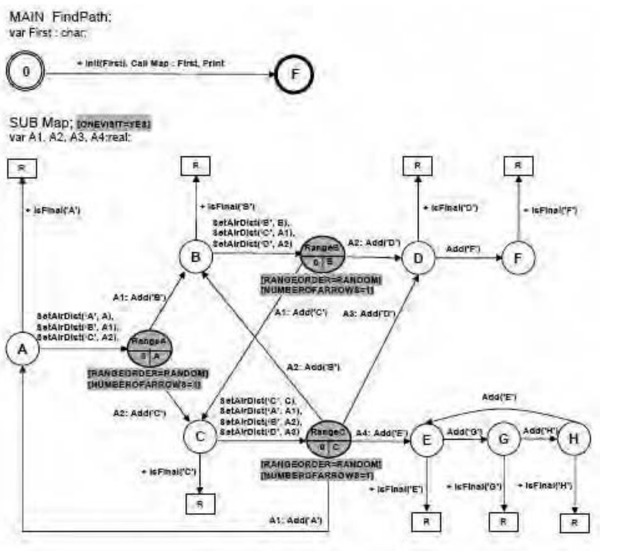


Figure 3 Stochastic hill climbing – nonprocedural non-recursive solution

Two CNP implementations of stochastic hill-climbing are described below on the example of the map traversal problem as defined in [8]. We assume that the map given is the one shown in Fig. 3 in [8]. As usual for similar problems, the heuristic used is the air distance between the cities. A third version was described in [8].

Fig. 3 illustrates a non-procedural solution in which a control state of type RANGE is used for every city in the map with multiple outgoing arrows – in our example these are cities A, B and C. Primitive SetAirDist sets the heuristic values of the arrows according to corresponding air distance. For example, the primitive call SetAirDist ('B',A1) associated with the arrow entering control state RangeA finds in a table the air distance from city B to the final city, and assigns this value to variable A1. The value of A1 is actually the heuristic value of the arrow from RangeA to B, in other words, the heuristic value of city B in the classical hill-climbing algorithm. Respectively, the value of variable A calculated by the primitive call SetAirDist('A',A) is the heuristic value of city A. Note that variable A is actually the higher selector, calculated dynamically, which determines the range for the control state RangeA and ensures movements downhill are not allowed. Option [RANGEORDER=RANDOM] must be used for each control state of type RANGE in subnet Map in order to set a random order of attempting the arrows outgoing from this state. The selection made is irrevocable due to [NUMBEROFARROWS=1]

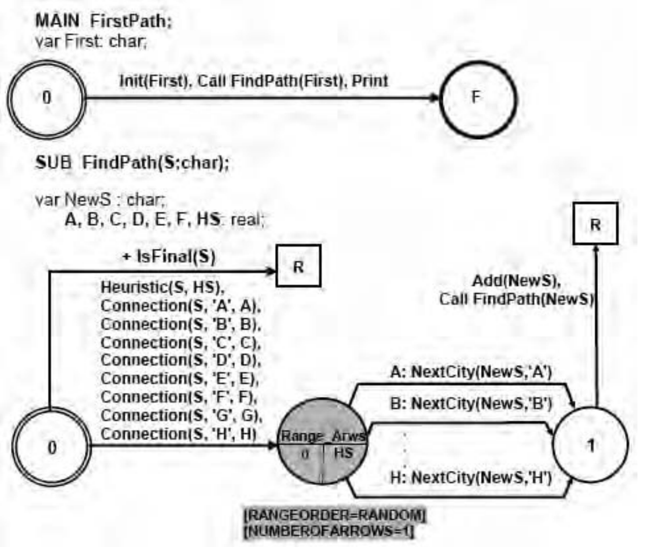


Figure 4 Stochastic hill climbing – nonprocedural recursive solution

Fig. 4 illustrates a recursive implementation of the hill-climbing strategy for the map traversal problem. Its idea is similar to the non-recursive solution shown on Fig. 3. Primitive Connection(City, NextCity, AirDist) checks if there is a road from City to NextCity. If such a connection exists then it sets the air distance from NextCity to the final city according to the table of air distances. In case no link exists the heuristic value is set to -1. The last value is outside the range and therefore such arrows will be pruned. Recursive solutions are typically preferable when the map (or the state space) is too large.

CNP realizations of the first-choice hill-climbing and simulated annealing algorithms are presented in [8].

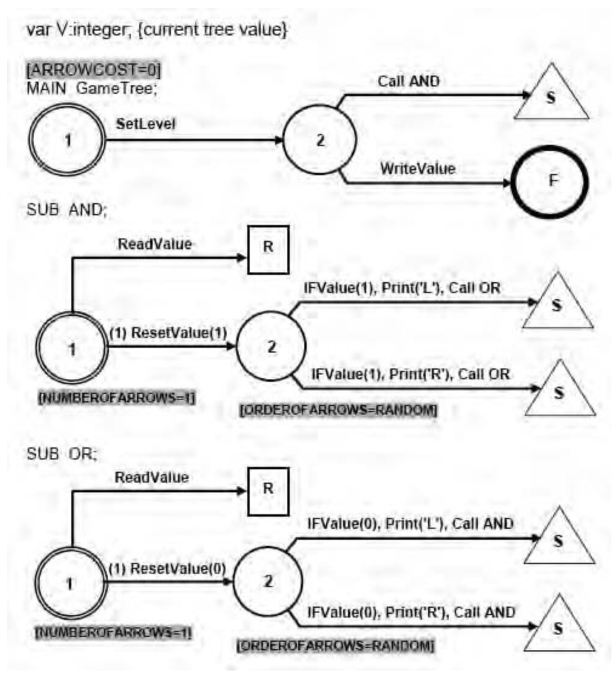


Figure 5 Game tree evaluation

TEACHING RANDOMIZATION IN ARTIFICIAL INTELLIGENCE – GAME TREE EVALUATION

Another illustration of using randomization is tree evaluation [16]. This example demonstrates a randomized algorithm whose expected running time is smaller than that of any deterministic algorithm.

We limit our discussion to the special case in which the possible values at the leaves are 0 or 1. Thus, each MIN node can be thought of as a Boolean AND operation and each MAX node as a Boolean OR operation. This special case is of interest in its own right, having applications in mechanical theorem proving. Fig. 5 depicts a CN implementing such a uniform binary tree evaluation.

To evaluate an AND node (through the subnet AND), one of its children (a sub-tree rooted at an OR node) is chosen at random (thanks to the system option [ORDEROFARROWS=RANDOM]) and it is evaluated by recursively invoking the algorithm. Only if the sub-tree value is 1 (controlled by the primitive IFValue), the algorithm proceeds to evaluate the other child (again by recursive application). If the value is 0, the algorithm returns 0 for the AND node. To evaluate an OR node (through the subnet OR), the procedure is the same with the roles of 0 and 1 interchanged.

CONCLUSION

Software tools in support of teaching and learning the concepts of randomized computation models and algorithms are almost non-existing. In particular, Prolog which is often used for modeling nondeterminism, is not applicable in the case of randomness.

We have shown that, on this background, CNP offers a convenient and effective educational environment and can be successfully used to help the students in their uneasy endeavor to apprehend and master the concept of randomization.

REFERENCES

[1] Kratchanov, K., Golemanova, E., Golemanov, T. and Külahçıoğlu, B.: Using Control Network Programming in Teaching Nondeterminism, this conference (2011)

[2] Wegener, I.: Teaching Nondeterminism as a Special Case of Randomization, Informatica Didactica, 4 (2001).

[3] Johnsonbaugh, R. and Schaefer, M.: Algorithms, Pearson – Prentice Hall (2004)

[4] Hromkoviĉ, J.: Algorithmics for Hard Problems: Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics, 2nd ed., Springer (2010)

[5] Rich, E.: Automata, Computability, and Complexity: Theory and Practice, Pearson – Prentice Hall (2008)

[6] Mandrioli, D. and Ghezzi, C.: Theoretical Foundations of Computer Science, Krieger Pub. Co. (1993)

[7] UPPAAL Pro: UPPAAL for Probabilistic Timed Automata: http://www.cs.aau.dk/~arild/uppaal-probabilistic/

[8] Kratchanov, K., Golemanov, T,, Golemanova, E and Ercan, T.: Control Network Programming with SPIDER: Dynamic Search Control, In: Knowledge-Based and Intelligent Information and Engineering Systems, Proc. 14th Intl Conf. (KES 2010), Cardiff, UK, Sep 2010, Part II, Lecture Notes in Artificial Intelligence), v.6277, Springer, pp. 253-262 (2010)

[9] Kratchanov, K., Golemanova, E., Golemanov, T. and Gökçen, Y.: Implementing Search Strategies in Winspider I: Introduction to Control Network Programming and Search. To appear in: Stanev, I. and K. Grigorova (eds.): Knowledge-Based Automated Software Engineering, Cambridge Scholars Publ. (2011)

[10] Kratchanov, K., Golemanova, E., Golemanov, T. and Ercan, T.. Non-Procedural Implementation of Local Heuristic Search in Control Network Programming, In: Knowledge-Based and Intelligent Information and Engineering Systems, Proc. 14th Intl Conf. (KES 2010), Cardiff, UK, Sep 2010, Part II, Lecture Notes in Artificial Intelligence), v.6277, Springer, pp. 263-272 (2010)

[11] Kratchanov, K., Golemanova, E., Golemanov, T. and Gökçen, Y.: Declarative and Procedural Search Strategy Implementations in Winspider, In: J. of the Technical University Sofia Branch Plovdiv "Fundamental Sciences and Applications", Vol. 16, pp. 217-222 (2011)

[12] Kratchanov, K., Golemanova, E, Golemanov, T. and Gökçen, Y.: Implementing Search Strategies in Winspider II: Declarative, Procedural, and Hybrid Approaches. To appear in: Stanev, I. and K. Grigorova (eds.): Knowledge-Based Automated Software Engineering, Cambridge Scholars Publ. (2011)

[13] Gurari, E.: An Introduction to the Theory of Computation, Computer Science Press (1989)

[14] Russell, S. and Norvig, P.: Artificial Intelligence: A Modern Approach, 3rd ed. Prentice Hall, Upper Saddle River, NJ (2010)

[15] Luger, G.F.: Artificial Intelligence: Structures and Strategies for Complex Problem Solving, 6th ed. Addison Wesley, Boston et al. (2009)

[16] Motwani, R. and Raghavan, P.: Randomized Algorithms, Cambridge University Press (1995)