ACTA TECHNICA CORVINIENSIS
BULLETIN OF ENGINEERING

[1.] IONEL MUSCALAGIU, [2.] DIANA MUSCALAGIU,
[3.] TEODORA PETRAS, [4.] MANUELA PĂNOIU

# DISCSP-NETLOGO-EDUCATIONAL SOFTWARE MEANT FOR THE IMPLEMENTATION AND EVALUATION OF THE ASYNCHRONOUS SEARCH TECHNIQUES IN NETLOGO

**Abstract:**
*The wide spreading of computer networks and of the Internet will result in the necessity of developing distributed software, which is supposed to work under these media, and to turn into account the advantages of a distributed and concurrent environment. The implementation of such techniques can be done in any programming language allowing a distributed programming, such as Java, by means of RMI. Nevertheless, for the study of such techniques, for the analysis of their completeness and for their evaluation, it is easier and more efficient to implement the techniques under certain distributed media, which offer various facilities, such as NetLogo.*
*In this article there is proposed a general implementation and evaluation model in NetLogo for the asynchronous techniques. This model, we believe, will allow the use of the NetLogo environment as a basic simulator for the study of asynchronous search techniques. This model can also be used for building educational software which can be used when studying the asynchronous search techniques with agents.*

**Keywords:**
*artificial intelligence, distributed programming, constraints, agents*

## INTRODUCTION

*The adjustment of the software technologies to the distributed equipment represents an important challenge for the next years. The wide spreading of computer networks and of the Internet will result in the necessity of developing distributed software, which is supposed to work under these media, and to turn into account the advantages of a distributed and concurrent environment.*
*The constraint programming is a model of the software technologies, used to describe and solve large classes of problems as, for instance, searching problems, combinatorial problems, planning problems, etc. A large variety of problems in the A.I field and other domains specific to computer sciences could be regarded as a special case of constraint programming. Lately, the A.I community showed a greater interest towards the distributed problems that are solvable through modeling by constraints and agents. The idea of sharing various parts of the problem between agents that act independently and that collaborate between them using messages, in the prospective of gaining the solution, proved itself useful, as it conducted to obtaining a new modeling type called Distributed Constraint Satisfaction Problem(DCSP) [3,4].*
*There exist complete asynchronous searching techniques for solving the DCSP, such as the ABT (Asynchronous Backtracking) and DisDB*

*(Distributed Dynamic Backtracking) [1,3,4]. There is also the AWCS (Asynchronous Weak-Commitment Search) [3,4] algorithm which records all the nogood values. The ABT algorithm has also been generalized by presenting a unifying framework, called ABT kernel [1]. From this kernel two major techniques ABT and DisDB can be obtained. The implementation of asynchronous search techniques based on distributed constraints can be done in any programming language allowing a distributed programming, such as Java, C, C++ or other. Nevertheless, for the study of such techniques, for their analysis and evaluation, it is easier and more efficient to implement the techniques under certain distributed environment, which offer various facilities, such as NetLogo [8]. [5,6,7].*

*NetLogo, is a programmable modelling environment, which canl be used for simulating certain natural and social phenomena. It offers a collection of complex modelling systems, developed in time. The models could give instructions to hundreds or thousands of independent agents which could all operate in parallel. NetLogo is the next generation in a series of modeling languages with agents that began with StarLogo [8]. It is an environment written entirely in Java, therefore it can be installed and activated on most of the important platforms (Windows, Unix) .*

*The aim of this article is to introduce an as general as possible model of implementation and evaluation for the asynchronous search techniques, in two possible cases: synchronous and asynchronous. This model can be used in the study of agents behavior in several situations, like the priority order of the agents, the synchronous and asynchronous case, leading, therefore, to identifying possible enhancements of the performances of asynchronous search techniques. This model can also be used in creating some educational software to be used in the study of asynchronous search techniques with agents by the students. For this purpose we have chosen the NetLogo environment, which is a programmable environment [8].*

*We will see the way one can simulate agents, how constraints can be implemented, how various measurement units for asynchronous techniques in the ABT and AWCS family can be implemented. Unfortunately, there is no distributed environment dedicated to modeling with distributed constraints, all the existent media are general ones, with more general targets. The implementation of agents and constraints implies a certain calculation effort, bigger or smaller, according to the performances of the given environment. The use of this support for educational software can ease the actual implementation of asynchronous techniques. This educational software ca be approached by students on the site [7]*

## THE IMPLEMENTATION OF APPLICATIONS WITH AGENTS IN NETLOGO

### THE NETLOGO OBJECTS

*The NetLogo world is made of agents. Each agent carries out a task, all the agents execute simultaneously and concurrently. The NetLogo language allows three types of agents: turtles, patches and the observer. The turtle type objects are agents that can move on in the NetLogo world, which is bidimensional and is divided in a grid of patches. Each patch is a square piece that represents the support on which turtle objects can move. The observer doesn't have a fixed location, it can be imagined as being situated above the world of turtles and patches objects. The observer can be regarded as a system agent that can initiate various operations for the other agents. NetLogo uses commands and reporters to tell the agents what to do (the commands and the reporters are NetLogo primitives). The commands are actions for the agents, but the reporters return certain values.*

*NetLogo allows the defining of different "types" of turtles, called breeds. Once a breed has been defined, we ca establish a different behavior for it. Those objects are used for simulating various objects existent in DCSP problems. For example, the agents from the n queens problem can be defined using breed type objects (a construction of type breeds [queens]). That thing allows the fixing of a special behavior for each agent-queen. When breed type objects are defined, automatically there is created an agentset for each breed.*

*A very important problem is related to the way of execution of an agent's attached procedures, agent simulated using breed type objects. The DCSP applications require the simultaneous and asynchronous execution of the code attached to each agent. That thing is possible in NetLogo*

*because the commands are executed asynchronously, each object of the "turtle" type or "patch" executes its list of commands as soon as possible. There are two ways of performing each agent's commands. The first one consists in "aligning" the commands executed by each turtle, through placing all the commands in the ask block. That way, the executed steps won't be synchronized. In exchange, using an ask command for each operation, a synchronization of all the operations performed by the agents will be obtained, each turtle will wait until the other turtle objects will finish their computations.*

## MODELLING AND IMPLEMENTING THE PROCESS OF THE AGENTS' EXECUTION

*In this paragraph there is presented a solution of modelling and implementation for the existing agents' process of execution in the case of the asynchronous search techniques. That modelling, applying a technique for detecting the algorithms' termination, allows us to obtain two multi-agent systems that can be applied for implementing and evaluating the most outstanding asynchronous search techniques. That modelling ca be used also for implementing most of the asynchronous search techniques, such as those from the AWCS family [3,4], ABT family [1], DisDB [1]. The modelling proposed in this paragraph allows the obtaining of implementations for asynchronous search techniques derived versions in which various situations that exist in reality are simulated: delays in supplying the messages, message management, etc. Implementation examples for those techniques can be found on the NetLogo site ([6] and in [5, 7]. Any implementation for the asynchronous search techniques supposes the following two steps:*

- *programming the agents such as they run concurrently*
- *designing the user interface.*

*The modelling of the agents' execution process will be structured on two levels, corresponding to the two stages of implementation. The definition of the way in which asynchronous techniques will be programmed such that the agents to run concurrently and asynchronous will be the internal level of the model. The second level refers to the way of representing the NetLogo application, and is the exterior level. The first aspect will be treated and represented using turtle type objects. The second aspect (that is connected with the problem to be solved) refers to the way of interacting with the user, the user interface. Regarding that aspect, NetLogo offers patches type objects de tip and various graphical controls. Anyway, patches type objects will allow the simulation of the application's interface.*

## AGENTS' SIMULATION AND INITIALIZATION

*First of all, the agents are represented by the breed type objects (as we saw in the previous paragraph, those are of the turtles type). In there fig. 1 is presented the way the agents are defined together with the global data structures proprietary to the agents.*

```
breeds [agents]
globals [variables that simulate the memory shared by all the agents]
agent-own [message-queue current-view nogoods messages-received_ok messages-received_nogood]
;message-queue contains the received messages.
;current-view is a list indexed on the agent's number, of the form [v0 v1 v2...], vi = -1 if we don't know the value o that agent.
;nogoods is the list of inconsistent positions [0 1 1 0 ... ] where 0 is a good position, and 1 is inconsistent.
;messages-received_ok and messages-received_nogood are variables that count the number of ok and nogood messages received by an agent.
```

*Figure 1. Agents' definition in the case of the asynchronous search techniques*

*The initialization of the agents supposes building the agents and initialization of the necessary data structures for the agents' operation. For initialization there is proposed an initialization procedure for each agent, procedure presented in figure 2 (the procedure will be called setup).*

```
to setup-agenţii // the agents defined with the breeds [agenţi] are used
; the num-agents agents are created and are initialized
  create-custom-agenţi num-agents [
    set messages-received_ok 0
    set current-view get-list num-agents -1
    set nogoods get-list num-agents 0
    set message-queue []
    ...
    ]
end
```

*Figure 2. The initialization procedure for each agent-setup*

*Typically the num-agents required for the running of the asynchronous search technique are built and the most important data structures are initialized.*

## REPRESENTATION AND MANIPULATION OF THE MESSAGES

*Any asynchronous search technique is based on the use by the agents of some messages for communicating various information needed for obtaining the solution. The manipulation of the messages supposes first of all message representation. This thing can be achieved in Netlogo by using some indexed lists. To represent complex messages that contain many information, Netlogo allows the use of lists of lists. The way of representation of the main messages encountered at the asynchronous search techniques is presented as follows:*

- *(**list** "ok" agent value agent_costs) – messages of the ok or info type;*
- *(**list** "nogood" agent current-view agent_costs) - messages of the nogood or back type;*
- *(**list** "addl" $agent_1$ $agent_2$ agent_costs)*
- *(**list** "removl" $agent_1$ $agent_2$ agent_costs )*

## DEFINITION AND REPRESENTATION OF THE INTERFACE

*As concerning the interface part, it can be used for the graphical representation of the DCSP problem's objects (queens or nodes) of the patch type. It is recommended to create an initialization procedure for the display surface where the agents' values will be displayed.*
*For the case of the graph coloring problem, the representation of nodes and links is done in the same way [5,6,7]. The two initialization procedures will be attached (using a setup procedure) to a start button of the application, as in the sequence in figure 3.*

```
to setup
  ca
  setup-patches
  setup-nodes
  ask nodes [procedura_inițializare]
end
```

*Figure 3. Setup procedure of the NetLogo application*

## IMPLEMENTATION AND EVALUATION METHODOLOGY FOR THE ASYNCHRONOUS TECHNIQUES

*In this paragraph there is presented a methodology of implementation for the asynchronous search techniques in NetLogo, using the model presented in the previous paragraphs 2. That methodology supposes the identification of the application's objects, building the agents and of the working surface for the application. There are also built the communication channels between agents, routines for message handling and the main program of the application. The methodology contains more elements specific to NetLogo necessary for finalizing the implementation of the asynchronous search techniques. Any implementation based on the presented model, will require the following of the next steps.*

***P1. Defining the DCSP application's objects.***

*Starting from the type of problem that is implemented, it will be defined the objects of the DCSP application. In figure 4 is presented a solution of agents modelling and also for the working surface of th eapplication. As in the modeling examples are proposed breeds [queens] (for modeling the agents associated to the queens from the problem of the n queens) or breeds [vertices] (for modeling the agents associated to each node from the problem of graph coloring).*

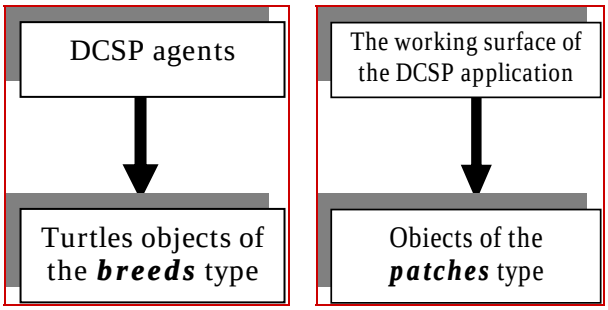
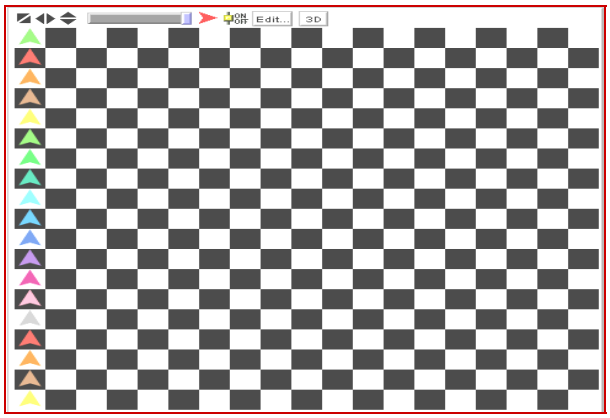
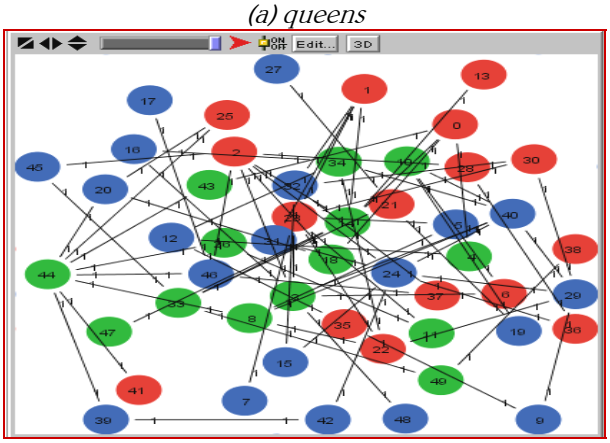



*Figure 4. Identification of the objects of the DCSP application*

*In exchange, to model the surface of the application are used objects of the patches type. Depending on the significance of those agents, they are represented on the Netlogo surface. In figure 5 are presented two ways in NetLogo for representing the agents of the queens type, respectively noduri.*



*(a) queens*

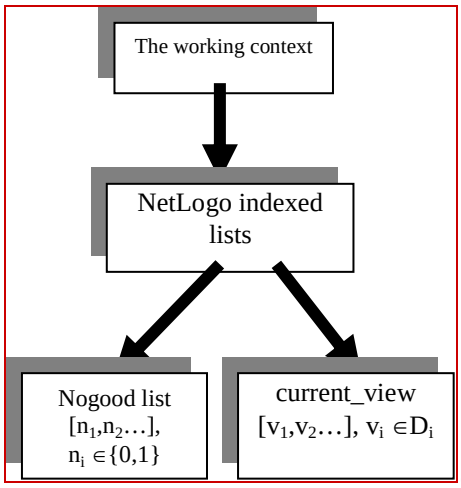


*(b) nodes*

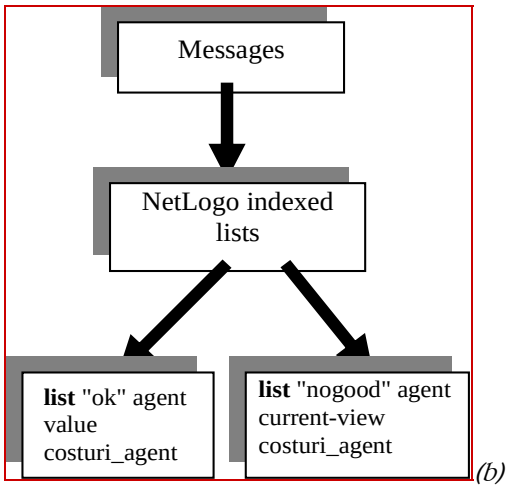
*Figure 5. Examples of reprezentation of the agents on the NetLogo surface*

***P2. Message handling. The FIFO type message chanel.***

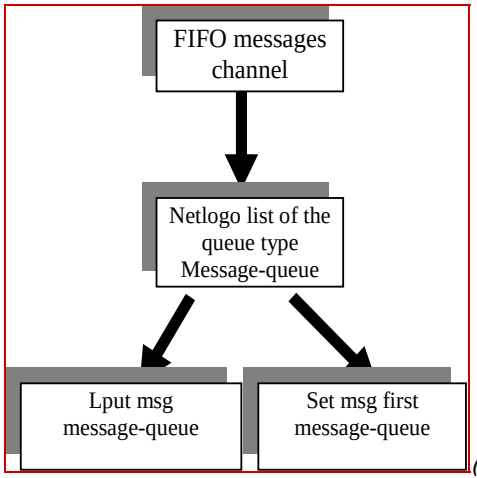
*Any agent keeps its working context at least as two proprietary structures: current_view and its nogood list. That context is used to take decisions, inclusively for building messages. For the proposed model, the data structures that store the working context of each agent can be simulated with lists. A representation solution is presented in figure 6 (a)*




*(a)*




*(b)*




*(c*

*Figure 6. The necessary structures for message handling)*

```
to receive-message
[msg]
 without-interruption
[
  set message-queue
lput msg message-
queue]
```

```
to-report retrieve-
message
  without-interruption [
   set msg first message-
queue
   set message-queue
butfirst message-queue]
 report msg
end
```

```
to handle-message
 if (empty?
message-queue)
[stop]
 set msg retrieve-
message
 if (first msg =
"ok")[
  ok msg ]
```

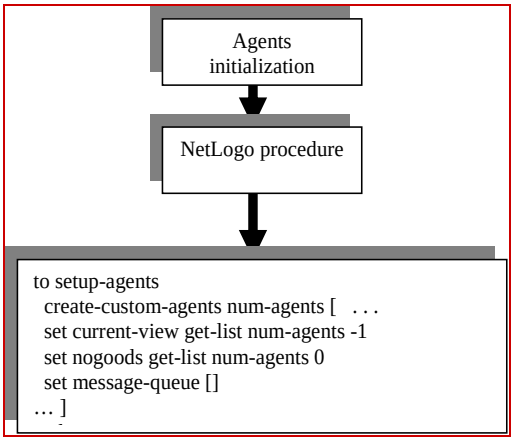
*Figure 7. Message handling*

*Message handlinf supposes first of all message representation. In figure 6.(b) is presented the way of representation of the main messages found at the asynchronous techniques. Simulareation of message queues for each agent can be done using Netlogo lists, for which are*

*defined routines of handling corresponding to FIFO principles (figure 6.(c)). These structures keep the messages received by each agent. Starting from NetLogo elements presented in figure 6, we can build three procedures for handling messages from the message queue, routines presented in figure 7. The first receive-message routine is used for receiving a new mesaage, the second routine retrieve-message has as its purpose the extraction of a message from the waiting queue, being called in the message treatment routine. The last routine handle-message identifies the message type, calling the appropriate message handling routine.*

***P3. Application initialization an of each agent. "The main program" for application***

*The initialization of the application supposes the building of agents and of the working surface for them. When the agents are built the required initializations are also done. Usually, is initialized the working context of the agentul (current-view), the message queues, the variables that count the effort carried out by the agent. In figure 8 are presented the two routines of application initialization and of agents initialization.*



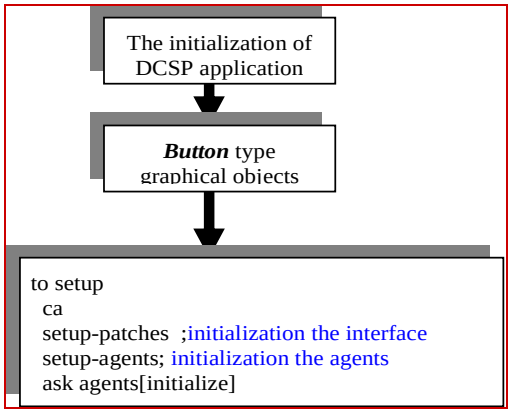




*Figure 8. Initialization of the DisCSP application*

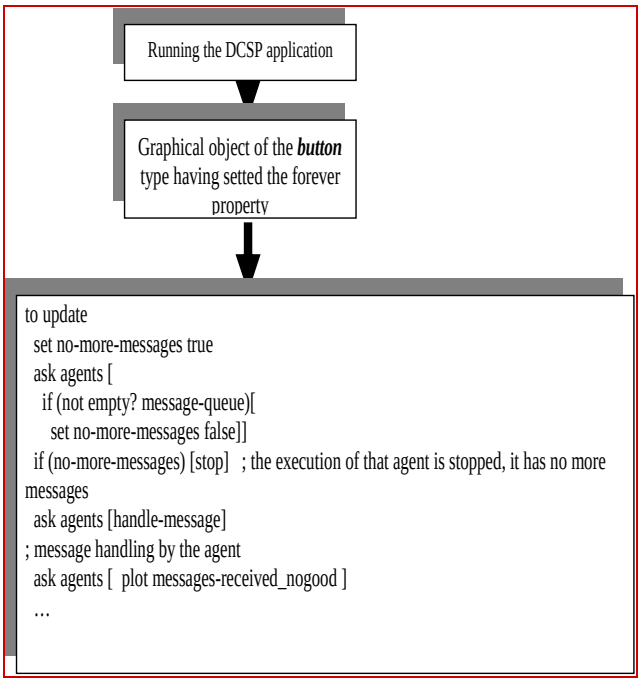



*Figure 9. The procedure for running the DCSP application for the system SEIS*

*The working surface of the application should contain NetLogo objects through whom the parameters of each problem could be controlled: the number of agents, the density of the constraints graph, the number of colors. These objects allow the definition and monitorization of each problem parameters.*

*For the application running is proposed the introduction of a graphical object of the button type and setting the forever property. That way, the attached code, in the form of a NetLogo procedure (that is applied on each agent) that will run continuously, until emptying the message queues and reaching the Stop command (which in NetLogo stops the execution of an agent). The solution presented in figure 9 is based on the utilization of the ask command. That NetLogo command executes a synchronization of each agent execution.*

*Another important observation is tied to attaching the graphical button to the observer. The use of this solution allows obtaining a solution of implementation with synchronization of the agents' execution. In that case, the observer will be the one that will initiate the stoping of the DisCSP application execution. In figure 9 the update procedure is attached and handled by the observer. These elements lead to the multi-agent system with synchronization of the agents execution (SEIS). If it's desired to obtain a system with asynchronous operation*

*(SIEAS), will be used the second method of detection, which supposes another update routine. That new update routine will be attached to a graphical object of the buton type which is attached and handled by the turtle type agents.*

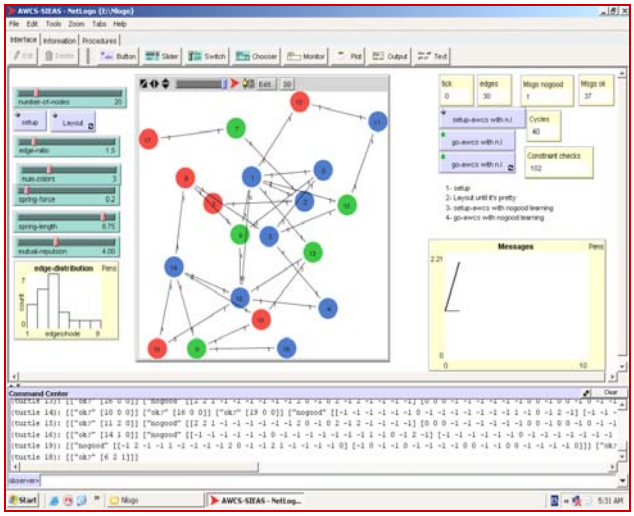


*Figure 10. NetLogo implementation for AWCS technique- SEIAS*

***P4. Monitorization of the evaluation parameters***

*The model presented in this chapter allows storing the costs for obtaining the solution. That thing can be done using some variables attached to the agents. For counting the flow of messages it can be used a variable proprietary to each agent (messages-received_nogood,etc), variable that needs to be incremented in the moment of receiving a message. That variable is incremented in the routine of message manipulation handle-message. Also, for measuring the work effort carried out by the agents can be used two variables nr_constraintc and c-ccks. Those variables store the costs necesary for each agent. Thus, those costs should be measured.*

*Application of the methodology presented previously allows the implementation and evaluation of any asynchronous search technique. In figure 10 is captured an implementation for the AWCS technique that uses uses the multi-agent SIEAS system.*

## CONCLUSION

*In this article was analysed the NetLogo environment with the purpose of building a general model of implementation and evaluation for the asynchronous techniques such as they could use the NetLogo environment as a basic simulator in the study of asynchronous search techniques.*

*In this article was proposed a general model of implementation and evaluation for the asynchronous search techniques. The proposed model supposed the identification of NetLogo objects necessary for implementing the asynchronous search technique (agents, messages, message queues, agents ordering) and of the interface of interaction with the user. In this article was proposed solutions for simulating the objects of any DisCSP application. Also, were proposed solutions for counting the costs for obtaining a solution using different measuring units. That thing will allow the evaluation of performances for asynchronous search techniques and eventual improvements for them Also, the model allows studying the behaviour of the agents for various techniques, studyind the costs for each agent.*

*As a general conclusion, we think that the model we achieved can be used for the study and analysis of the asynchronous techniques, the model allowing their complete evaluation. Students can use the models on the site [7] to study, to understand the functioning of the asynchronous search techniques and, perhaps, to extend them. Starting from those models, they can develop other versions of the asynchronous search techniques.*

## REFERENCES


[1.] C. BESSIERE, I. BRITO, A. MAESTRE, P. MESEGUER, Asynchronous Backtracking without Adding Links: A New Member in the ABT Family. Artificial Intelligence, 161:7-24, 2005

[2.] MUSCALAGIU, H. JIANG, H.E. POPA. "Implementation and evaluation model for the asynchronous techniques: from a synchronously distributed system to a asynchronous distributed system", in Proceedings of the 8th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2006), Timisoara, Romania, IEEE Computer Society Press, 2006, pp. (209-216)

[3.] YOKOO M., E.H. DURFEE, T. ISHIDA, K. KUWABARA (1998): The distributed constraint satisfaction problem: formalization and algorithms. IEEE Transactions on Knowledge and Data Engineering 10 (5)

[4.] YOKOO MAKOTO (2001): Distributed Constraint Satisfaction- Foundation of Cooperation in Multi-agent Systems. Springer
[5.] MAS Netlogo Models-a. http://jmvidal.cse.sc.edu/netlogomas/
[6.] MAS Netlogo Models-b. http://ccl.northwestern.edu/netlogo/models/community
[7.] MAS Netlogo Models-c. http://discsp-netlogo.fih.upt.ro/
[8.] WILENSKY, U. NetLogo itself: NetLogo. http://ccl.northwestern.edu/ netlogo/. Center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, IL, 1999

## AUTHORS & AFFILIATION

[1.] IONEL MUSCALAGIU,
[2.] DIANA MUSCALAGIU,
[3.] TEODORA PETRAS,
[4.] MANUELA PĂNOIU

[1. 2. 3. 4.] UNIVERSITY „POLITEHNICA" TIMISOARA, FACULTY OF ENGINEERING HUNEDOARA, ROMANIA