Abstract

The use of agent-based simulation models in NetLogo for research is growing rapidly in a number of fields of science and engineering, including in computer science, for example for analyzing the performances of asynchronous search techniques based on distributed constraints. Distributed Constraint programming (DisCSP/DCOP) is a programming approach used to describe and solve large classes of problems such as searching, combinatorial and planning problems. The purpose of this paper is to present an open-source and complete solution in NetLogo that allows modeling, simulation and evaluation of the distributed constraints using single computer or clusters, simulation or real execution. Our tool allows the use of various search techniques and also the evaluation and analysis of the performance of the asynchronous search techniques in two manners: simulation mode and real execution. We also explain our methodology for running the NetLogo models in a cluster computing environment (simulation mode) or on a network of computers (simulation real execution). This tool is aimed to allow the evaluation of distributed algorithms in conditions as similar as possible to the real situations.

Key words: Distributed Constraint programming, distributed modeling, NetLogo, asynchronous search techniques.

1. Introduction

Agent-based modeling is a type of modeling in which the focus is on representing agents and their interactions. It has gained extensive use in the fields of economics, social science, ecology, biology and other scientific domains.

Constraint programming is a software technology used to describe and solve large classes of problems such as searching, combinatorial and planning problems. A Distributed Constraint Satisfaction Problem (DisCSP) is a constraint satisfaction problem in which variables and constraints are distributed among multiple agents. A Distributed Constraint Optimization Problem (DCOP) is similar to the constraint satisfaction problem except that the constraints return a real number instead of a Boolean value and the goal is to minimize the value of these constraint violations. This type of distributed modeling appeared naturally for many problems for which the information was distributed to many agents. DisCSPs are composed of agents, each owning its local constraint network. Variables from different

E-mail address: ionel.muscalagiu@fih.upt.ro
agents are connected by constraints forming a network of constraints. Agents must assign values to their variables so that all constraints between agents are satisfied. Instead, for DCOP a group of agents must distribute values for a set of variables so that the cost of a set of constraints over the variables is either minimized or maximized.

Many problems in the areas of computer science, engineering, biology can be modeled as constraint satisfaction problems (or distributed CSP). Some examples include: spatial and temporal planning, diagnosis, decision support, hardware design and verification, real-time systems and robot planning, protein structure prediction problem, DNA structure analysis, timetabling for hospitals, industry scheduling, transport problems\(^2\), etc. Many studies regarding distributed constraints were made, they proved the success of constraint modeling.

There are some algorithms for performing distributed search in cooperative multiagent systems where each agent has some local information and together with the other agents agents will find the problem’s solution. There are many search algorithms in these networks of constraints for solving the DisCSP (called asynchronous searching techniques), such as the ABT (Asynchronous Backtracking)\(^{13}\), ABTDO (Dynamic Ordering for Asynchronous Backtracking)\(^7\), DisDB (Distributed Dynamic Backtracking)\(^3\), etc. Also, for DCOP there are many algorithms among whom we name ADOPT (Asynchronous Distributed OPTimization)\(^8\) or DPOP (Dynamic Programming Optimization Protocol)\(^11\). We find that many multiagent problems can be reduced to a distributed constraints problem.

The implementation and testing of the asynchronous search techniques implies a programming effort not at all trivial. Thus, the necessity of developing a dedicated platform that can be used for testing them became a necessity. There are some platforms for implementing and solving DisCSP problems: DisChoco\(^{16}\), DCOPolis\(^{12}\), FRODO\(^5\) and DisCSP-Netlogo\(^{10,18}\). Such a tool allows the use of various search techniques so that we can decide which is the most suitable one for that particular problem. Also, these tools can be used for the study of agents’ behavior in several situations, like the priority order of the agents, the synchronous and asynchronous case, apparition of delays in message transmission, therefore leading to identifying possible enhancements of the performances of asynchronous search techniques.

The asynchronous search techniques involves concurrent (distributed) programming. The agents can be processes residing on a single computer or on several computers, distributed within a network. The implementation of asynchronous techniques can be done in any programming language allowing a distributed programming (such as Java) or other such as the new generation of multiagent modeling language (NetLogo\(^{14,18,17}\)).

NetLogo is regarded as one of the most complete and successful agent simulation platforms\(^6,14\). NetLogo is a high-level platform, providing a simple yet powerful programming language, built-in graphical interfaces and the necessary experiment visualization tools for quick development of simulation user interface. It offers a collection of complex modeling systems, developed in time. It is an environment written entirely in Java, therefore it can be installed and activated on most of the important platforms (Linux, Windows, Mac OS X). NetLogo has been enhanced with new capabilities. We focus on HubNet\(^{15}\), a NetLogo extension based on a client-server architecture useful for participatory simulations, used in teaching a class, the teacher can run these activities with participants, and discuss learning experiences afforded by these activities. Such a capability is used in this article to extend the DisCSP-Netlogo models for running them in a manner as close as possible to a real situation.

The DisCSP-NetLogo modules were designed for the implementation, learning and evaluation of distributed algorithms\(^8,10\). These modules offer support for researchers from the DisCSP/DCOP domain for developing their algorithms, for evaluating the performances of these algorithms, and why not, for tutorials that allow teaching these techniques that have a pretty high degree of difficulty. Also, these modules allow the study of the agents’ behavior, visualization of various values attached to the agents and, as a consequence, allowing us to understand these complex systems.

We adapted the HubNet technology to allow the agents to run on computers or mobile devices from the local network or over the Internet. A template model is provided, this template can be downloaded from the websites\(^{18}\). This thing will allow us to run the agents in conditions as close as possible to the real ones, opposite to the majority of dedicated platforms that allow an evaluation in the simulated mode. Typically, the DisCSP/DCOP algorithms have been evaluated along two dimensions: computation and communication. Many authors have also introduced delays in simulated message communication, previous studies showing that delays influence the performances of the algorithms. The solution proposed here will allow to recreate more realistic scenarios and to better understand algorithms behavior in conditions of existing network latency.
Our goal is to supply the programmer with sources that can be updated and developed so that each can take profit from the experience of those before them. The proposed approach (the modules that compose the DisCSP-Netlogo platform) is open-source and can be used for implementing any search technique. Any researcher has access to the existing implementations and can start from these for developing new ones. This paper extends the capabilities of the DisCSP-Netlogo platform, where the agents were simulated in a distributed environment, to a real execution in a network of computers.

In this article are proposed two extensions for making the running closer to a real one: running with as many as possible agents (solution of cluster computing, using some minimal NetLogo-core branch models that allows approximately 1000 agents) and extending the models for running them in a network. So, using the HubNet technology, is proposed the extending of the NetLogo models so that the agents could run on other computers or mobile devices (Android, iOS, etc.).

This paper is organized as follows. Section 2 presents the old solution for modeling and simulation in NetLogo of the distributed constraints. That solution runs also in a cluster of computers and is extended by adding some minimal NetLogo-core branch modules. In section 3 is presented the new solution which extends the old one to the real-execution mode of agents in distributed constraints. In Section 3, we explain our methodology for running the NetLogo models in the real-execution mode. Section 4 presents briefly a discussion on the facilities offered by the NetLogo platform. The conclusions of the paper are in Section 5.

2. Modeling and implementing of the asynchronous search techniques in NetLogo

In this section we present a solution of modeling and implementation for the existing agents’ process of execution in the case of the asynchronous search techniques. This open-source solution, called DisCSP-NetLogo will be extended in the next section so that it is able to run a real-execution model, runnable in a local network of computers or on a number of mobile devices (tablets, phones).

This modeling can also be used for any of the asynchronous search techniques, such as those from the AWCS family, ABT family, DisDB, etc. Implementation examples for these techniques can be found on the site in. This modeling approach in NetLogo was presented in.

The modeling of the agents’ execution process is structured on two levels, corresponding to the two stages of implementation. The definition of the way in which asynchronous techniques are programmed so that the agents run concurrently and asynchronously constitutes the internal level of the model. The second level refers to the way of representing the surface of the implemented applications. This is the exterior level.

In any NetLogo agent simulation, four entities (objects) participate:

- **The Observer**, that is responsible for simulation initialization and control. This is a central agent.
- **Patches**, i.e. components of a user defined static grid (world) that is a 2D or 3D world, which is inhabited by turtles. Patches are useful in describing environment behavior.
- **Turtles** that are agents that “live” and interact in the world formed by patches. Turtles are organized in breeds, that are user defined groups sharing some characteristics, such as shape, but most importantly breed specific user defined variables that hold the agents’ state.
- **Links** agents that “connect” two turtles representing usually a spatial/logical relation between them.

The definition of turtle specific variables allows them to carry their own state and facilitates the encoding of complex behavior. Agents’ behavior can be specified by the domain specific NetLogo programming language, that supports functions (called reporters) and procedures. The language includes a large variety of primitives for turtles motion, environment inspection, etc. More details can be found on the NetLogo site.

2.1. Agents’ simulation and initialization

First of all, the agents are represented by the breed type objects (those are of the turtles type). In Fig. 1 is presented the way the agents are defined together with the global data structures proprietary to the agents. We implement in open-source NetLogo the agents’ process of execution in the case of the asynchronous search techniques.
S1. Agents’ simulation and initialization in DisCSP-NetLogo. First of all, the agents are represented by the breed type objects (those are of the turtles type). Fig. 1 shows 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 MyValue Nogoods nr-constraintc messages-received-ok messages-received-nogood AgentC-Cost]
   Message-queue contains the received messages.
   :Current-view is a list indexed on the agent’s number, of the form [v0 v1...],
   :vi = -1 if we don’t know the value of that agent.
   :nogoods is the list of inconsistent value [0 1 1 0 ... ], where 1 is inconsistent.
   :messages-received-ok, etc, count the number of messages received by an agent.
   :nr-cycles -the number of cycles, nr-constraintc - the number of constraints checked
   :AgentC-Cost - a number of non-concurrent constraint checks
```

This type of simulation can be applied for different problems used at evaluation and testing:
- the distributed problem of the n queens characterized by the number of queens.
- the distributed problem of coloring of a randomly generated graph characterized by the number of nodes, colors and the number of connections between the nodes.
- the randomly generated (binary) CSPs characterized by the 4-tuple (n, m,p1,p2), where: n is the number of variables; m is the uniform domain size; p1 is the portion of the n · (n − 1)/2 possible constraints in the constraint graph; p2 is the portion of the m · m value pairs in each constraint that are disallowed by the constraint.
- the randomly generated problem that has a structure of scale-free network (the constraint graph has a structure of scale-free network).
- the multi-robot exploration problem.

For these types of problems used in the evaluation there are NetLogo modules that can be included in the future implementations. The modules are available on the website. This modules can generate problem instances (both solvable and unsolvable problems) with various structures for the previous problems, depending on various parameters. An example is presented in Fig. 2 for the random binary problems.

```
breeds [agents-nodes]
breeds [edges]
   :nodes = agents, each undirected edge goes from a to b
   :edges = links agents that connect two agents-nodes representing usually a spatial/logical relation between them.
globals [Orders done nr-cycles domain-colour-list no-more-messages]
agent-own [Message-queue Current-view MyValue Nogoods Neighbours-list ChildrenA messages-received-ok AgentC-Cost ... ]
```

Fig. 1. Agents’ definition in DisCSP-Netlogo for the asynchronous search techniques

S2. 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 agents’ communication is done according to the communication model introduced in.

The communication model existing in the DisCSP frame supposes first of all the existence of some channels for communication, of the FIFO type, that can store the messages received by each agent. The way of representation of the main messages is presented as follows:

```
to setup : Setup the model for a run, build a constraints graph.
   setup-global ; setup Global Variables
   setup-patches ; initialize the work surface on which the agents move
   setup-turtles we generate the objects of the turtles type that simulate the agents
   setup-binary-problems ; or LoadRBRFile
   ; we generate a the types of problems used at the evaluation or we load an instance generated and saved previously.
   CalculateOrdersMaxCardinality ; is selected from variable-ordering heuristics
   setup-DisCSP //we initialize the data structures necessary for the DisCSP algorithm
   end
```

Fig. 2. Templates for agents’ definition in DisCSP-Netlogo for the the random binary problems
The simulation of the message queues for each agent can be done using Netlogo lists, for whom we define treatment routines corresponding to the FIFO principles. These data structures are defined in the same time with the definition of the agents. In the proposed implementations from this paper, that structure will be called message-queue. This structure property of each agent will contain all the messages received by that agent.

The manipulation of these channels can be managed by a central agent (which in NetLogo is called observer) or by the agents themselves. In this purpose we propose the building of a procedure called go for global manipulation of the message channels. It will also have a role in detecting the termination of the asynchronous search techniques' execution. That go procedure is some kind of a “main program”, a command center for agents. The procedure should also allow the management of the messages that are transmitted by the agents. It needs to call for each agent another procedure which will treat each message according to its type. This procedure will be called handle-message, and will be used to handle messages specific to each asynchronous search technique.

**S3. Definition and representation of the user interface.**

As concerning the interface part, it can be used for the graphical representation of the DisCSP problem’s objects (agents, nodes, queens, amino acids, robots, obstacle, link, etc.) of the patch type. It is recommended to create an initialization procedure for the display surface where the agents’ values will be displayed.

![NetLogo implementation for the protein folding problem](image1)

![2D square lattice representation for the multi-robot exploration problem](image2)

**Fig. 3. Representation of the environment in the case two problems**

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 Fig. 3 is presented the manner in which the agents are represented in NetLogo.

**S4. Running the DisCSP problems.**

The initialization of the application supposes the building of agents and of the working surface for them. Usually are initialized the working context of the agent, the message queues, etc. The working surface of the application should contain NetLogo objects through whom the parameters of each problem could be controlled in real time: the number of agents (nodes, robots), the density of the constraints graph, etc. For launching the simulation is proposed the introduction of a graphical object of the button type and setting the forever property. Another important observation is tied to attaching the graphical button to the observer. The use of this approach allows obtaining a solution of implementation with synchronization of the agents’ execution. In that case, the observer agent will be the one that will initiate the stopping of the DisCSP algorithm execution. These elements lead to the multiagent system with synchronization of the agents’ execution. If it’s desired to obtain a system with asynchronous operation, the second method of detection will be used, which supposes another update routine. That new go routine will be attached to a graphical object of the button type which is attached and handled by the turtle type agents.

In Fig. 4 is captured an implementation of the ABT for the random binary problems technique that uses the model presented. The update procedure is attached and handled by the turtle type agents (Fig. 4). Implementation examples for the ABT family, DisDB, DBS, ADOPT and the AWCS family can be downloaded from the website. More implementation details of the DisCSP/DCOP in Netlogo are available as a tutorial from.
Another important thing that can be achieved in NetLogo is related to the evaluation of the asynchronous algorithms. The model presented within this paper and in previous studies allows the monitoring of the various types of metrics: the number of messages transmitted during the search, the number of cycles, the number of constraints checked, a number of non-concurrent constraint checks and other. The models presented allow real-time visualization of metrics. During runtime, using graphic controls, various metrics are displayed and updated in real-time, after each computing cycle.

3. Enhancing DisCSP-NetLogo from simulation to real-execution of agents in distributed constraints

3.1. The architecture of the multi-agent system

HubNet is a technology that lets you use NetLogo to run participatory simulations in the classroom. In a participatory simulation, a whole class takes part in enacting the behavior of a system as each student controls a part of the system by using an individual device, such as a networked computer or mobile device (tablets and phones with Android, etc).

HubNet is a new architecture designed to give students the experience of participating as elements in a simulation of a complex dynamic system. It is based on a client-server architecture, where HubNet hardware includes an upstream computer (server, the “hub”) capable of addressing a network of nodes (currently, networked computers, mobile devices with Android or TI-83+ calculators) and a display capability (e.g., computer projection system) enabling an entire class to view the simulation. HubNet enables many users at the “nodes” to control the behavior of individual objects or agents and to view the aggregated results on a joint display.

The software architecture of HubNet contains an NetLogo application called Computer HubNet (HubNet server) and many instances of the client application. Computer HubNet is regarded as a main server and uses networked computers as nodes. The HubNet architecture can support other devices as nodes (mobile device with Android). The network layer implements flexible communication protocols that include the ability to upload and download data sets, upload and download programs, support real-time interaction as in network computer games, and form collaborative groups of various sizes (e.g., peer to peer, small groups, and whole-class modes). The activity leader uses the NetLogo application to run a HubNet activity (HubNet server). Participants use a client application to log in and interact with the HubNet server.

Starting from the HubNet architecture we propose a new model that allows modeling and running various search algorithms. The basic idea is to move the agents to run on each network node (computer or mobile device) as opposed to the simulator mode, where they are simulated and run on the same computer. We will adapt this architecture to allow distributed running of the agents in the network. In Fig. 5 is presented the HubNet architecture.

In this architecture can be remarked the two entities of the distributed application. That is, the central application - HubServer and the client applications - HubNetClient. The central application (HubNet server) contains the definitions.
3.2. A methodology of implementation and evaluation for the asynchronous search techniques in DisCSP-NetLogo in the real-execution of agents mode

In this paragraph is presented a methodology of implementation for the asynchronous search techniques in NetLogo, using the model presented in the previous paragraph. That methodology supposes the identification of the two entities of the distributed application: the central application - HubServer and the client applications - HubNetClient. Any implementation based on the presented model, will require us to follow the next steps:

**S1.** Create a NetLogo model according to the previous model for the asynchronous search techniques and for the types of problems used at the evaluation. First, the NetLogo model will require an initialization stage. That includes the interface initialization, initialization of the network module, the activation of HubNet clients and generating the agents. The proposed solution supposes for procedures called `setup`, `login-clients`, `GenerateProblems` and `setup-DisCSP`. At a minimum it will need the following lines of code in Fig. 6.

```netlogo
; The Setup Procedure in DisCSP-NetLogo
to setup ; Setup the model for a run, build a constraints graph.
  setup-globals ; setup Global V ariables
  setup-patches ; initialize the surface of the application
  are used objects of the patches type
  hubnet-reset; we initialize the network mode, which will ask
  ;the user for a session name and
  ;open up the HubNet Control Center
  ...
end

; The login procedure

; The GenerateProblems Procedure in DisCSP-Netlogo

; The setup-algorithm procedure
```

**S2.** Next, all models must also have a `go` (update) procedure. The wrapper runs the NetLogo program by asking it to loop for a certain number of times and allows the finalizing of the DisCSP algorithm.
Usually for the DisCSP algorithms, the solution is generally detected only after a break period in sending messages (this means there is no message being transmitted, state called quiescence). In such a procedure, that needs to run continuously (until emptying the message queues) for each agent, the message queue is verified (to detect a possible break in message transmission). In the case of the extended solution in which the agents run as HubNet clients, the checking is done by each agent, but the informations are transmitted to the central application (HubNet server) which decides (in the case that all the queues are empty) that a state called quiescence is reached.

Sample code for the go procedure in the case of asynchronous search techniques can be found in Fig. 7.

```go
// The running procedure
every 0.1 [
    set no-more-messages true
    ; get commands and data from the clients- listen-to-clients -as long as there are more messages from the clients keep processing them.
    while [ hubnet-message-waiting? ] [ [hubnet-fetch-message ; get the first message in the queue
    ifelse hubnet-enter-message? ; The clients send messages when it login
        [hubnet-send hubnet-message-source "Accept-agent" "No"]
        [if hubnet-exit-message? ; The clients send messages when it logout
            [Show "Error - too few agents" ; stop]
            [Process-Queue hubnet-message-tag ]
        ]
    ]
    ask-concurrent agents [
        if (not empty? message-queue) [set no-more-messages false]
    ]
    ifelse (no-more-messages and Not done) [WriteSolution : hubnet-broadcast "Solution" "Yes" ; stop]
    [if (done)
        [show "No solution" : hubnet-broadcast "Solution" "No solution" ; stop]
    end]
]
```

Fig. 7. The Go Procedure in DisCSP-Netlogo for the asynchronous search techniques with synchronization of the agents’ execution

Another observation, each HubNet client signals the reception of a message and transmits to the HubNet server application that thing, for the latter to run the agent’s code. The solution with HubNet clients (of the NetLogo type) doesn’t allow running the code effectively by the clients. In a ulterior version we will extend the HubNet clients functionality so that they will run entirely the code (using Java modules that communicate using sockets).

Another observation, the HubNet clients don’t allow direct transmission (peer-to-peer) of the messages. They are brokered by the central HubNet server application.

The procedure should also allow the management of messages that are transmitted by the agents. For that, when a HubNet client receives a message, it needs to call another procedure (that is called handle-message) and is used to handle messages specific to each asynchronous search technique. The handling of the communication channels will be performed by this central agent. These elements will lead to a variant of implementation in which the synchronizing of the agents’ execution is done.

The distributed application’s work flow is composed of the following steps:

S1. Start the HubNet Server. It is done by pressing the initialization button setup. The runtime parameters are set: number of variables, their domains, the constraints graph density (p1-network-connectivity), etc. The button will initialize the network support.

S2. Activating the connections with the clients. For that press the login button on the computer with the HubNet server to allow the clients to connect. On each client computer is launched the clients manager (HubNet Control Center). Using that tool the HubNet Clients are opened, a username is chosen and connect to the main activity (to HubNet Server). When all the HubNet clients have connected the login button is disabled.

S3. An instance for the evaluated problem is generated. For that the button GenerateProblems is pressed. Optional, the Layout button can be used, to redraw the surface on the screen, until we consider it to be pretty.

S4. The button setup-DisCSP is activated. That will initialize the data structures necessary for the DisCSP/DCOP algorithm.

S5. The main application is runned (on the HubServer) using the go button.

S6. Finally, the measurements and the problem’s solution are collected.

In Fig. 8 are presented two captures for the two entities, HubNetServer and HubNetClient.
4. Discussion

There are very few platforms for implementing and solving DisCSP problems: DisChoco\textsuperscript{16}, DCOPolis\textsuperscript{12} and FRODO\textsuperscript{3}. It is interesting to see if the proposed platform brings some benefits compared to other platforms. The solution presented in this paper, based on NetLogo, has these features:

- the modules can be adapted and personalized for each algorithm. There is a very large community of NetLogo users that can help for development.
- it allows the communication between agents, without being necessary to call directly the communication system (it is independent of the network support);
- the models can allow the simulation of multiagent systems on a single machine, and also on a cluster;
- DisCSP-NetLogo provides a special agent Observer, that is responsible for control interface. The AgentObserver allows the user to track operations of a DisCSP algorithm during its execution.
- there are facilities such as agentsets that allow the implementation of agents that manage more variables.
- manipulating large quantities of information requires the use of databases, for example for nogood management, using the SQL extension of NetLogo we can store and access values from databases.
- NetLogo allows users to write new commands in Java and use them in their models (using extensions).

Another discussion refers to the advantages of running NetLogo models on a cluster of computers in the variants with complete NetLogo or with minimal install (core branch).

In most of the articles about DisCSP/DCOP, the evaluations of the algorithms are made for maximum 100 agents. The cluster allowed running instances over 500 agents, with various difficulties (even 1000 but with a lower difficulty)\textsuperscript{9}. It is interesting to see what is the effect of running the models on a cluster of computers, if the runtime is reduced for the analyzed techniques. For that, we performed some empirical tests with a variable number of agents (n=500 and n=1000). We examined the performance of AWCS in scale-free networks (we implemented and generated in NetLogo both solvable and unsolvable problems that have a structure of scale-free networks). Scale-free networks are generated with the following parameters: nodes = 500, |D| = 10, md = 4 and \( \gamma = 1.8 \), respectively nodes = 1000, \( \text{md} = 4 \) and \( \gamma = 2.1 \).

For the evaluations, we generate five scale-free networks. For each network, the constraint tightness is fixed at 0.4 and 100 random problem instances are generated. The runs were performed in three variants: on a single computer using the model with GUI (C1), on a single computer, but without GUI (headless (C2)) and on the Infragrid cluster (C3). For each was counted the total amount of time for running 100 instances. The results are the following: Nodes=1000, C1–8h,53min, C2–3h,8min and C3–38min, respectively Nodes=500, C1–47min, C2–16min and C3–5min.

The analysis of the results shows that the solution running on a cluster of computers allows problems with big dimensions for the addressed problems, and also a short runtime.
5. Conclusions

In this paper we introduce a model extended for the study and evaluation of the asynchronous search techniques in NetLogo using the typical problems used for evaluation, model called DisCSP-NetLogo.

An open-source solution for implementation and evaluation of the asynchronous search techniques in NetLogo is presented, model that can be run in a network of computers or on mobile devices. The solution proposed here will allow to recreate more realistic scenarios and to better understand algorithms behavior in conditions of network latency.

In this paper we have developed a methodology to run NetLogo models in a network of computers, on mobile devices or on a single machine, varying both parameter values and/or random number of agents. The GUI solution can to be runned on a network of computers in the mode with synchronization. The open-source solution presented in this paper can be used as an alternative for testing the asynchronous search techniques, in parallel with the platforms already consecrated as DisChoco, DCOPolis, FRODO, etc.

The solution presented here supplies support for learning and studying the asynchronous search techniques. Each participant (through a client application or applet) can control an agent or an agentset in the same simulated world, created and managed by a server.

Future research will include running more sets of experiments with two large families: the ABT family and the AWCS family applied to the typical evaluation problems (the distributed problem of the m-coloring of a randomly generated graph, the multi-robot exploration problem, the random binary CSPs). In the future we wish to extend the clients so that they are completely independent, using Java modules that communicate using sockets. Also, future work will involve creating a web-based version of the client.

Acknowledgments

This work was partially supported by the European Commission grant FP7-REGPOT-CT-2011-284595 (HOST) and by the Romanian national grant PNII-ID-PCE-2011-3-0260 (AMICAS).

References