Large Scale Multiagent-Based Simulation using NetLogo for implementation and evaluation of the distributed constraints

Ionel Muscalagiu¹, Popa Horia Emil² and Jose Vidal³

¹ The Faculty of Engineering of Hunedoara, The "Politehnica" University of Timisoara, Revolutiei, 5, Romania, ionel.muscalagiu@fih.upt.ro
² The Faculty of Mathematics and Informatics, The University of the West, Timisoara, V.Parvan 4, Romania hpopa@info.uvt.ro
³ Computer Science and Engineering, University of South Carolina, Columbia SC 29208, USA vidal@sc.edu

Abstract. 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. This type of distributed modeling appears naturally in many problems for which the information is distributed among many agents. Modeling and simulation are essential tools in many areas of science and engineering, including computer science. The purpose of this paper is to present an open-source solution for the implementation and evaluation of the distributed constraints in NetLogo using computer clusters. Our tool allows the use of various search techniques and also the evaluation and analysis of the performance of the asynchronous search techniques. We also explain our methodology for running the NetLogo models in a cluster computing environment or on a single machine, varying both parameter values and/or the random number of agents.

1 Introduction

Most multiagent systems are characterized by a set of autonomous agents each with local information and ability to perform an action when the set of actions of all must be coordinated in order to achieve a desired global behavior. Constraint programming is a programming approach 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 [15], [7]. 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.
Distributed Constraint Satisfaction/Distributed Constraint Optimization is a framework for describing a problem in terms of constraints that are known and enforced by distinct participants (agents). The constraints are described on some variables with predefined domains, and have to be assigned to the same values by the different agents. 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 in different 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 distributedly choose values for a set of variables so that the cost of a set of constraints over the variables is either minimized or maximized. Distributed networks of constraints have proven their success in modeling real problems.

There are some algorithms for performing distributed search in cooperative multiagent systems where each agent has some local information and where the goal is to get all the agents to set themselves to a state such that the set of states in the system is optimal. There exist complete asynchronous searching techniques for solving the DisCSP in this constraints network, such as the ABT (Asynchronous Backtracking), AWCS (Asynchronous Weak Commitment) [15], ABTDO (Dynamic Ordering for Asynchronous Backtracking) [7], AAS (Asynchronous Search with Aggregations) [13], DisDB (Distributed Dynamic Backtracking) [2] and DBS (Distributed Backtracking with Sessions) [9]. 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. Thus, the asynchronous searching techniques for solving the the DisCSP have many different applications.

Developing of evaluation and testing tools for the search techniques became a necessity. There are very few platforms for implementing and solving DisCSP problems: DisChoco [17], DCOPolis [12] and FRODO [4]. 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 any asynchronous search techniques supposes building the agents and the existing constraints, the implementation of the links between the agents and the communication channels between them. The implementation of asynchronous search techniques 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 a certain distributed environment, such as the new generation of multiagent modeling language (NetLogo [16], [19], [20], [5]).

NetLogo is regarded as one of the most complete and successful agent simulation platforms [16], [5]. 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. The models could give instructions to hundreds or thousands of independent agents which could all operate in parallel. It is a environment written entirely in Java, therefore it can be installed and activated on most of the important platforms. Although, excellent for “modeling social and emergent phenomena”, i.e. agent based simulations that consist of a large number of reactive agents, it lacks the facilities to model easily more complex goal oriented agent behaviors.

Modeling and simulation is an essential tool in many areas of science and engineering, including in computer science, for example, for analyzing the performances of asynchronous search techniques. The asynchronous search techniques can be remarked by the existence of a very large number of elements that can be introduced, without affecting the completeness of the algorithm. For example, processing the message in packets or individual, storage or not of the nogood messages, message filtering, nogood learning and storing them for each value. Each run is affected by delays appeared in message transmission. Thus, a correct evaluation assumes a large number of runs, with different data sets. The purpose of this paper is to present an open-source solution for implementation and evaluation of the asynchronous search techniques in NetLogo, for a great number of agents, model that can be run on a cluster of computers. However, 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, etc.

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. In fact, it is the idea of development adopted in the Linux operating system. In this paper is proposed such an implementation and evaluation solution for existing algorithms in the DisCSP/DCOP framework. The proposed approach is open-source and can used for implementing any search technique. Any researcher has access to the existing implementations and can start from these for developing new ones. The platform offers modules for various problems of evaluation such as: the random binary problems, random graph coloring, multi-robot exploration.

This paper synthesizes all the tries of modeling and implementation in NetLogo for the asynchronous search techniques. Many preliminary studies were done and published on the modeling, implementation and evaluation of the asynchronous search techniques in NetLogo. Many implementations were done for a class of algorithms from the ABT and AWCS families (DisCSP), respectively ADOPT (DCOP). They can be downloaded from the websites [19], [20].

This paper is organized as follows. Section 2 presents a solution for modeling and simulation in Netlogo of the distributed constraints. In Section 3, we explain
our methodology for running the NetLogo models in a cluster computing environment. In this section is presented an architecture for the multiagent system. 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 is extended so that it is able to run on a larger number of agents, model runnable on a cluster of computers and is presented below. This modeling can also be used for any of the asynchronous search techniques, such as those from the AWCS family [15], ABT family [2], DisDB [2], DBS [9]. Implementation examples for these techniques can be found on the sites in [19],[20]. This modeling approach in NetLogo was first presented in [10] in a preliminary form.

The modeling of the agents' execution process is structured on two levels, corresponding to the two stages of implementation [10], [18]. 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 initialisation 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 organised 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.

Both patches, turtles and links carry their own internal state, stored in a set of system and user-defined variables local to each agent. 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.
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 Figure 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 [10],[18]:

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). Figure 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 values [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
```

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

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 \( \cdot (n-1)/2 \) possible constraints in the constraint graph; p2 is the portion of the m \( \cdot m \) 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) [1]. An instance DisCSP that has a structure of scale-free network have a number of variables with a fixed domain and are characterized by the 5-tuple (n, m, t, md, \( \gamma \)), where n is the number of variables, m is the domain size of each variable; t (the constraint tightness) determining the proportion of value combinations forbidden by each constraint, md=the minimal degree of each nodes and \( \gamma \) is the exponent that depends on each network structure. A scale-free network is characterized by a power-law degree distribution as follows \( p(k) \propto k^{-\gamma} \) [1].
- **the multi-robot exploration problem** [3] are characterized by the 6-tuple (n, m, p1, sr, cr, obsd), where:
– \( n \) is the number of robots exploring an environment, interact and communicate with their spatial neighbors and share a few common information (information about already explored areas);

– \( m = 8 \) is the domain size of each variable; \( \text{Dom}(x_i) \) is the set of all 8 cardinal directions that a robot \( A_i \) can choose to plan its next movement.

– \( p_1 \) - network-connectivity, \( s_r \) - the sensor range of a robot, \( c_r \) - the communication range of a robot;

– \( \text{obsd} \) - obstacles-density. We have considered environments with different levels of complexity depending on: the number of obstacles, the size of the obstacles, the density of the obstacles.

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 [19]. For each module are available procedures for random generation of instances for the choosen problems, together with many ways of static ordering of the agents. Also, there are procedures for saving in files the generated instances and reusing them for the implementation of other asynchronous search techniques. On the website [19] can be found many modules that can generate problem instances (both solvable and unsolvable problems) with various structures for the previous problems, depending on various parameters (uniform random binary DisCSP generator, scale-free network instance generator for DisCSP). An example is presented in Figure 2 for the random binary problems.

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 [15].

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:

\[
\text{list } \text{"type message" contents Agent-costs} \quad ;
\]

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 \texttt{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 \texttt{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.
DisCSP-Netlogo - an open-source tool in NetLogo for distributed constraints

breeds [agents-nodes]
breeds [edges]
:agents-nodes = agents, each undirected edge goes from a to b
:edges = links agents that connect two agents-nodes

globals [Orders done nr-cycles domain-colour-list no-more-messages]
agent-own [Message-queue Current-view MyValue Nogoods Neighbours-list ChildrenA
ParentA nr-constraintc messages-received-ok messages-received-nogood AgentC-Cost]

includes "RBP.nls" "StaticOrders.nls"

Fig. 2. Templates for agents’ definition in DisCSP-Netlogo for the random binary problems

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, 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.

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 3 is presented the way in NetLogo for representing the agents.

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, the variables that count the effort carried out by the agent. 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. These objects allow the definition and monitoring of each problem’s parameters.

For launching the simulation 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) will run con-
Another important observation is tied to attaching the graphical button to the observer [10]. 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 (the go procedure is attached and handled by the observer). 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 [10], [19]. 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.

![Fig. 4. NetLogo's graphical interface and code tab](image)

Fig. 4. NetLogo implementation of the ABT with temporary links for the random binary problems, n=100 agents

In Figure 4 is captured an implementation of the ABT with temporary links for the random binary problems technique that uses the model presented. The update procedure is attached and handled by the turtle type agents (Figure 4). These elements lead to a multiagent system with agents handling asynchronously the messages. Implementation examples for the ABT family, DisDB, DBS and the AWCS family can be downloaded from the website [19].

More details of implementation of the DisCSP/DCOP in Netlogo are not presented here but are available as a tutorial and downloadable software from [19].

### 2.2 The evaluation of the asynchronous search techniques

Another important thing that can be achieved in NetLogo is related to the evaluation of the asynchronous algorithms. The model presented within this paper allows the monitoring of the various types of metrics:

- the number of messages transmitted during the search: messages-received-ok, messages-received-nogood, messages-received-nogood-obsolete, etc.
- the number of cycles. A cycle consists of the necessary activities that all the agents need in order to read the incoming messages, to execute their local calculations and send messages to the corresponding agents. This metrics allows the evaluation of the global effort for a certain technique
the number of constraints checked. The time complexity can be also evaluated by using the total number of constraints verified by each agent. It is a measurement of the global time consumed by the agents involved. It allows the evaluation of the local effort of each agent. The number of constraints verified by each agent can be monitored using the variables proprietary to each robot called $nr_{\text{constraint}}$.

- a number of non-concurrent constraint checks. This can be done by introducing a variable proprietary to each agent, called $AgentC_{\text{Cost}}$. This will hold the number of the constraints concurrent for the agent. This value is sent to the agents to which it is connected. Each agent, when receiving a message that contains a value $SenderC_{\text{Cost}}$, will update its own monitor $AgentC_{\text{Cost}}$ with the new value.

- the total traveled distance by the robots. This metric is specific to the multirobot exploration problem [3]. It makes it possible to evaluate if an algorithm is effective for mobile and located agents in an unknown environment.

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. Also, the metrics’ evolution can be represented as graphics using plot-like constructions (models from [19] include some templates).

3 Running on a Linux cluster

In this paragraph we will present a methodology to run the proposed NetLogo models in a cluster computing environment or on a single machine. We utilize the Java API of NetLogo as well as LoadLeveler. LoadLeveler is a job scheduler written by IBM, to control scheduling of batch jobs. This solution is not restricted to operate only in this configuration, it can be used on any cluster with Java support and it operates with other job schedulers as well (such as Condor).

Such a solution will allow running a large number of agents (nodes, variables, robots, queens, etc.). The first tests allowed running of as much as 500 agents, in the conditions of a high density constraint graph. The first experiments were done on the InfraGrid cluster from the UVT HPC Centre [21], on 100 computing systems (an hybrid x86 and NVIDIA Tesla based). InfraGRID is an Linux only cluster based on a mixture of RedHat Enterprise Linux 6 and CentOS 6. For Workload Management, JOB execution is managed at the lowest level by IBM LoadLeveler.

The methodology proposed in the previous paragraph that uses the GUI interface will run on a single computer. In this paragraph we will present a new solution, without GUI, that can run on a single computer or on a cluster.

The proposed approach uses the NetLogo model presented previously, runnable without the GUI, with many modifications. In order to run the model in that manner is used a tool named BehaviorSpace, existent in NetLogo. BehaviorSpace is a software tool integrated with NetLogo that allows you to perform experiments with models in the "headless" mode, that is, from the command line,
without any graphical user interface (GUI). This is useful for automating runs on a single machine, and can also be used for running on a cluster.

BehaviorSpace runs a model many times, systematically varying the model’s settings and recording the results of each model’s run. Using this tool we develop an experiment that can be run on a single computer (with a small number of agents) or, in the headless mode on a cluster (with a large number of agents).

We will now present the methodology for creating such an experiment [6]. The steps necessary for the implementation of a multiagent system are as follows:

**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. For running it on the cluster and without GUI some adaptations have to be made. First, the NetLogo model must have a procedure called setup to instantiate the model and to prepare the output files. At a minimum it will need the following lines of code in Figure 5.

```plaintext
; Setup the model for a run, build a constraints graph.
setup ; setup-globals ; 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-random-problem ; we generate the types of problems used at the evaluation.
setup-DisCSP ; we initialize the data structures necessary for the DisCSP algorithm
end
```

Fig. 5. The Setup Procedure in DisCSP-Netlogo.

Next, all models must also have a **go** (update) procedure. The **go** procedure is a bit different than the usual NetLogo program. 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). This situation can be resolved by checking the message queues, queues that need to be empty. 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 transmitting).

The procedure should also allow the management of messages that are transmitted by the agents. The procedure needs to call for each agent another procedure (that is called handle-message) and is used to handle messages specific to each asynchronous search technique. The two procedures are the most important from the point of view of their messages handling way asynchronous or synchronous (way of work that defines the asynchronous techniques).

The first solution of termination detection is based on some of the facilities of the NetLogo: the **ask-concurrent** command that allows the execution of the computations for each agent and the existence of the central observer agent. 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
to go // The running procedure
set no-more-messages true
set nr-cycles nr-cycles + 1
ask-concurrent agents |
  if (not empty? message-queue)[
    set no-more-messages false]]
if (no-more-messages) [WriteSolution stop]
ask-concurrent agents [handle-message]
end

Fig. 6. The Go Procedure in DisCSP-Netlogo for the asynchronous search techniques with synchronization of the agents’ execution

Synchronizing of the agents’ execution is done. Sample code for the go procedure in the case of asynchronous search techniques can be found in Figure 6.

S2. Create an experiment using BehaviorSpace and parse the NetLogo file into an input XML file (so that it can be runned in the headless mode, that is without GUI). In Figure 7 is presented a simple example of XML file.

```
<experiments>
  <experiment name="experiment" repetitions="10">
    <setup>setup</setup>
    <go>go-mrp</go>
    <final>WriteMetrics</final>
    <exitCondition>Final</exitCondition>
    <enumeratedValueSet variable="p1-network-connectivity">
      <value value="0.2"/>
    </enumeratedValueSet>
    ...
  </experiment>
</experiments>
```

Fig. 7. The XML file for the multi-robot exploration problem

To finalize the run and adding up the results it is recommended the use of a Netlogo reporter and a routine that writes the results. The run stops if this reporter becomes true.

```
java -Xmx1024m # use up to 1GB RAM
-Dfile.encoding=UT F8 # for compatible cross-platform
-classpath NetLogo.jar # specify main jar
org.nlogo.headless.Main # specify that we want headless, not GUI
--model NetLogo -- Model.nlogo # the Netlogo model that runs
--experiment name = of -- experiment # the name of the experiment
```

Fig. 8. The script for multiple runs on a cluster for DisCSP

S3. Create a Linux shell script (in sh or in bash) that describes the job for LoadLeveler. Once the Netlogo model is completed with the experiment created
with the BehaviorSpace tool, it is time to prepare the system for multiple runs. To do this, first create a script that allows running with no GUI, an example script is presented in Figure 8.

![Architecture of a multiagent system with synchronization of the agents' execution](image)

**Fig. 9.** Architecture of a multiagent system with synchronization of the agents’ execution

In Figure 9 is presented this multiagent system’s architecture for running on a cluster of computers.

### 4 Discussion

There are very few platforms for implementing and solving DisCSP problems: DisChoco [17], DCOPolis [12] and FRODO [4]. In [17] a DisCSP/DCOP platform should have the following features:

- be reliable and modular, so it is easy to personalize and extend;
- be independent from the communication system;
- allow the simulation of multiagent systems on a single machine;
- make it easy to implement a real distributed framework;
- allow the design of agents with local constraint networks.

The solution presented in this paper, based on NetLogo, has these features:

- the modules can be adapted and personalised 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 simulation initialisation and control interface. The AgentObserver allows the user to track operations of a DisCSP algorithm during its execution. Also, there are 4 tools (Globals Monitor, Turtle Monitor, Patch Monitor and Link Monitor) that allow monitoring of global variables values, the values of the variables associated to the agents.
- 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 and reporters in Java and use them in their models (using extensions).

5 Conclusions

In this paper we introduce an model of study and evaluation for 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, for a great number of agents, model that can be run on a cluster of computers is presented. Such a tool allows the use of various search techniques so that we can decide on the most suitable one.

In this paper we have developed a methodology to run NetLogo models in a cluster computing environment or on a single machine, varying both parameter values and/or random number of robots. We utilize the Java API of NetLogo as well as LoadLeveler. The solution without GUI allows to be run on a cluster of computers in the mode with synchronization, as opposed to the GUI solution that can be runned on a single computer and allows running in both ways: with synchronization or completely asynchronously.

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.

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). Also, building modules for integrating MySQL and PostgreSQL databases for nogood management.

References