DisCSP-Netlogo- an open-source framework in NetLogo for implementation and evaluation of the distributed constraints

Ionel Muscalagiu,[1] Popa Horia Emil[2] and Jose Vidal [3]

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. A simulation framework in NetLogo for distributed constraints search and optimization algorithms is presented. The purpose of this paper is to present an open-source solution for the implementation and evaluation of the distributed constraints in NetLogo. This tool can run with or without a graphical user interface in a cluster of computers with a large number of agents. It includes all needed techniques for implementing all existing DCSP and DCOP algorithms. A comparison with the main evaluation and testing platforms for distributed constraints search and optimization algorithms is presented.

1 Introduction

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 [16], [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). 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) [16], ABTDO (Dynamic Ordering for Asynchronous Backtracking) [7], AAS (Asynchronous Search with Aggregations) [14], 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) [12]. We find that many multiagent problems can be reduced to a distributed constraints problem. 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, etc.

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 [19], DCOPolis [13], DisCo [4], FRODO [5] and DisCSP-Netlogo [10, 21]. 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 search techniques can be done in any programming language allowing a distributed programming, such as Java, C/MPI 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 [17], [21], [22], [6]).

NetLogo is regarded as one of the most complete and successful agent simulation platforms [17], [6]. NetLogo is a high-level platform, providing a simple yet powerful programming language, built-in graphical interfaces and the necessary experiment visualization

[1] Politehnica University of Timisoara, Romania, email: mionel@.fih.upt.ro.
[2] The University of the West, Timisoara, Romania, email:hpopa@info.uvt.ro
[3] Computer Science and Engineering, University of South Carolina, USA, email:vidal@sc.edu

tools for quick development of simulation user interface. It is a environment written entirely in Java, therefore it can be installed and activated on most of the important platforms.

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. 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 [10],[11]. 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 [21], [22].

The DisCSP-NetLogo modules were designed for the implementation, learning and evaluation of distributed algorithms [10, 11]. 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.

The DisCSP Netlogo allows various extensions: the modules can be updated and extended. The test problem generators can be particularized and new problem types can be added. Extreme situations can be simulated, such as network delays, allowing users to test the same algorithms under different network conditions. The DisCSP Netlogo allows the running with GUI also the visualization in real time of the metrics associated with the algorithm. The visualization of partial solutions is also possible with this tool. By visualization of various aspects of distributed search algorithms, one can attain new insights on the behavior of these algorithms. These insights may help drive new variants of search algorithms and different heuristics for these algorithms. The visualization also enables algorithm debugging during the implementation process of new algorithms, and can serve as an educational method which dynamically displays an algorithms progress.

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 [21]. 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. The solution proposed here will allow to recreate more realistic scenarios and to better understand algorithms behavior in conditions of existing network latency. This tool is aimed to allow the evaluation of distributed algorithms in conditions as similar as possible to the real situations.

Some problems modeled with distributed constraints are exemplified:

- 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.
- the randomly generated (binary) CSPs.
- the protein folding problem.

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 [16], ABT family [2], DisDB [2], DBS [9]. Implementation examples for these techniques can be found on the sites in [21],[22].

The modeling of the agents' execution process is structured on two levels, corresponding to the two stages of implementation [10], [20]. 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.

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],[20]:

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.

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.

```
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 ... 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 ... ], where 1 is inconsistent.
;messages-received-ok,count the number of messages received by an agent.
;nr-cycles -the number of cycles,
;AgentC-Cost - a number of non-concurrent constraint checks
```

Figure 1. Agents' definition in DisCSP-Netlogo for the asynchronous search techniques

-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, γ), 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 γ 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; Dom(x_i)is the set of all 8 cardinal directions that a robot A_i can choose to plan its next movement.
- $p1$ - network-connectivity, sr - the sensor range of a robot, cr - the communication range of a robot;
- $obsd$ - 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 [21]. 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 [21] 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 [16].

```
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 ChildrenA
ParentA nr-constraintc messages-received-ok ... AgentC-Cost]

__includes["RBP.nls""StaticOrders.nls"]
;are included the modules for generating instances and choosing
a static order for the agents ...
to setup ; Setup the model for a run, build a constraints graph.
  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-binary-problems ; or LoadRBRFile
  ; we generate a the types of problems used at the evaluation
  ; or we load an instance generated and salved previously.
  CalculateOrdersMaxCardinality ; is selected from variable-ordering heuristics
  setup-DisCSP
;we initialize the data structures necessary for the DisCSP algorithm
end
```

Figure 2. Templates for agents' definition in DisCSP-Netlogo for the random binary problems

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:

- (list "type message" *contents* Agent-costs) ;

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.

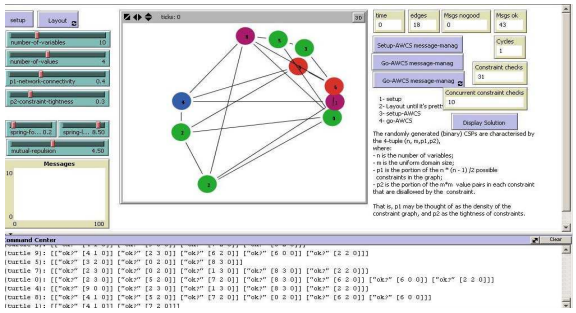
The manipulation of these channels can be managed by a central agent (which in NetLogo is called observer) or by the agents themselves. In this purpose we propose the building of a procedure called *go* for global manipulation of the message channels. It will also have a role in detecting the termination 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.

2.2 The Graphical User Interface

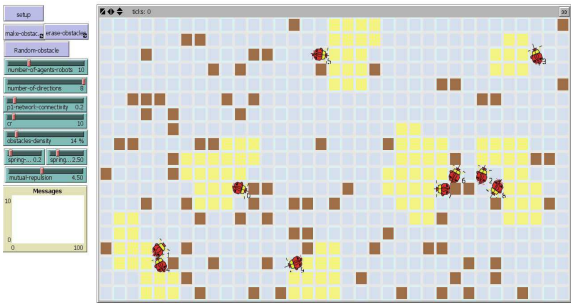
S3. Definition and representation of the user interface.

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, 4 is presented the way in NetLogo for representing the agents.

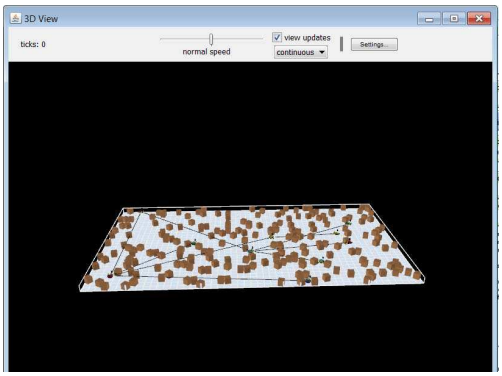


(a) Netlogo representation for the graph coloring problem

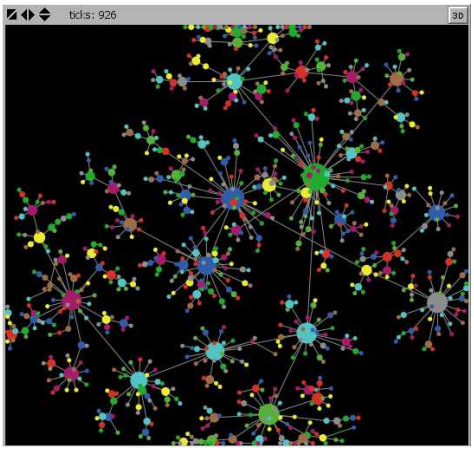


(b) The 2D square lattice representation for the multi-robot exploration problem

Figure 3. Representation of the environment in the case three problems

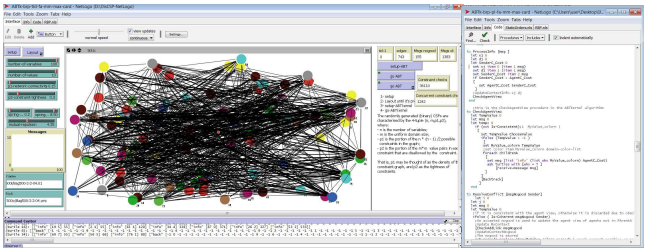


(a) The 3D square lattice representation



(b) A scale-free network with 900 nodes

Figure 4. Representation of the environment in the case three problems



(a) NetLogo's graphical interface (b) NetLogo's code tab

Figure 5. NetLogo implementation of the ABT with temporary links for the random binary problems, n=100 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 which 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 continuously, until emptying the message queues and reaching the stop command. 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], [21]. 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 Figure 5 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 5). 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 [21].

More details of implementation of the DisCSP/DCOP in Netlogo are not presented here but are available as a tutorial and downloadable software from [21].

2.3 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 pre-

sented 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. These 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 agents called $nr - constraintc$.
- a number of non-concurrent constraint checks. This can be done by introducing a variable proprietary to each agent, called $AgentC - Cost$. 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 $SenderC - Cost$, will update its own monitor $AgentC - Cost$ with the new value.
- the total traveled distance by the robots. This metric is specific to the multi-robot 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 [21] include some templates).

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 [18], 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). It is based on a client-server architecture, where HubNet hardware includes an up-front 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 [17, 18] 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, support real-time interaction as in network computer games, etc [18]. 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. 6 is presented the HubNet architecture.

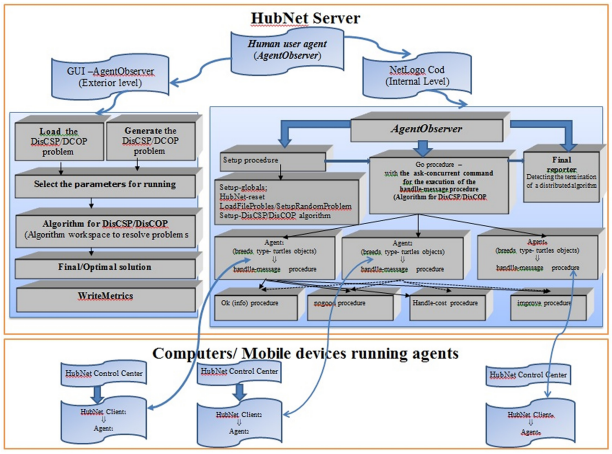


Figure 6. The DisCSP architecture for simulating and real-execution of agents in distributed constraints

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 of the agents and the procedure for handling the communication chanels. The HubNet clients will be attached to each agent.

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*. At a minimum it will need the following lines of code in Fig. 7.

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.

```
to setup ; Setup the model for a run, build a constraints graph.
  setup-globals ; setup Global Variables
  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
```

(a) The Setup Procedure in DisCSP-Netlogo

```
to login-clients ; allows agents-HubNet Client to log into the
;activity without running the model or collecting data
  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
  [ create-new-agents
  hubnet-send agent-id "Accept-agent" "Yes"]
  [ ..]
  ]end
```

(b) The login procedure

```
to GenerateProblems ; Build a constraints graph.
  setup-random-problem ; we generate the types of problems
used at the evaluation.
  CalculateOrders; is selected from variable-ordering heuristics
  ...
end
```

(c) The GenerateProblems Procedure in DisCSP-Netlogo

Figure 7. Initialization of the multi agent system.

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 transmited 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. 8.

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:

```
to 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 the
    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
```

Figure 8. The Go Procedure in DisCSP-Netlogo for the asynchronous search techniques with synchronization of the agents' execution

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 run (on the HubServer) using the *go* button.

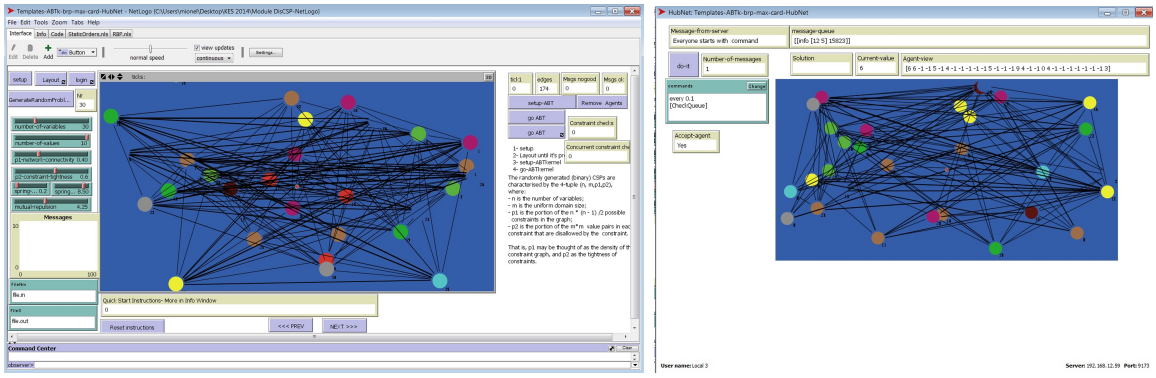
S6. Finally, the measurements and the problem's solution are collected.

In Fig. 9 are presented two captures for the two entities, HubNet-Server and HubNetClient.

4 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 [23], 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



(a) The HubNetServer (b) The HubNetClient

Figure 9. NetLogo implementation of the ABT -HubNet version

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 runned 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. 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 10.

```
to setup ; Setup the model for a run, build a constraints graph.
  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
```

Figure 10. The Setup Procedure in DisCSP-Netlogo.

Next, all models must also have a *go* (update) procedure. 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.

```
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
```

Figure 11. The Go Procedure in DisCSP-Netlogo for the asynchronous search techniques with synchronization of the agents' execution

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 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 11.

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 12 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"/>
   ...
   < /experiment>
< /experiments>
```

Figure 12. 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.

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.

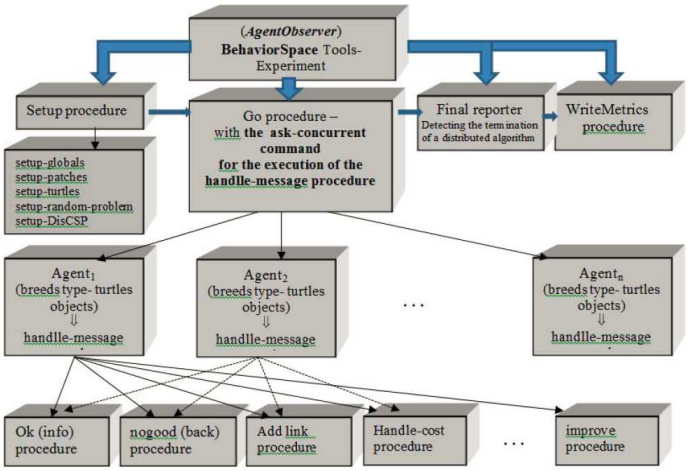


Figure 13. Architecture of a multiagent system with synchronization of the agents' execution

In Figure 13 is presented this multiagent system's architecture for running on a cluster of computers.

5 Discussion

There are very few platforms for implementing and solving DisCSP problems: DisChoco [19], DCOPolis [13] and FRODO [5]. In [19] 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.

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 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 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)[10]. 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, $|D_i| = 10$, md = 4 and $\gamma = 1.8$, respectively nodes = 1000, 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 folowing: 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. The platform based on NetLogo has the following differences with the other platforms:

1. The sources of the algorithms are accessible, one can obtain implementations derived from the DisCSP/DCOP algorithms.
2. The programmer has complete access to the algorithms code and can intervene.
3. The possibility to run on a cluster of computers allows the simulation of problems with thousand or tens of thousands agents. In most of the studies performed, one can remark that most of the tests are done on problems with a small number of agents (below 100).

In Table 1 are presented the difference between platforms for implementing and solving DisCSP/DCOP problems.

6 CONCLUSION

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 presented a methodology to run NetLogo models in a cluster computing environment or on a single machine, varying

Table 1. The difference between platforms for implementing and solving DisCSP/DCOP problems

The platform	Support for implementing the DisCSP algorithms	Support for implementing DCOP algorithms	Support for real running in an distributed environment	Access to algorithm sources	Implementation of most of the DisCSP algorithms	Implementation of most of the DCOP algorithms	Being able to run on a cluster
DisCo	Yes	Yes	No	No	Yes	Yes	No
DisChoco	Yes	Yes	Yes	Yes	partial	partial	Yes
DCOPolis	No	Yes	Yes	No	No	Yes	?
FRODO	No	Yes	Yes	No	No	Yes	Yes
DiCSP-Netlogo	Yes	Yes	Yes	Yes	Yes	partial	Yes

both parameter values and/or random number of agents. 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 run 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. A comparison with the main evaluation and testing platforms for distributed constraints search and optimization algorithms is presented.

As future developments, we will try to implement the modules on the HPC Repast platform, in a Logo-like C++. This thing will allow running on supercomputers with a very high number of agents (above 100,000 agents).

REFERENCES

[1] A. L. Barabasi and A. L Albert, *Emergence of scaling in random networks*, Science, 286 (1999), p. 509-512.

[2] C. Bessiere, I. Brito, A. Maestre, P. Meseguer. *Asynchronous Backtracking without Adding Links: A New Member in the ABT Family*. A.I., 161:7-24, 2005.

[3] A. Doniec, N. Bouraqadi, M. Defoort, V-T Le, and S. Stinckwich. *Multi-robot exploration under communication constraint: a disCSP approach*. In 5th National Conference on "Control Architecture of Robots", May 18-19, 2010 - Douai, FRANCE.

[4] A. Grubshtein, N. Herschhorn, A. Netzer, G. Rapaport, G. Yafe and A. Meisels. *The Distributed Constraints (DisCo) Simulation Tool*. Proceedings of the IJCAI11 Workshop on Distributed Constraint Reasoning (DCR11), pages 30–42, Barcelona, 2011.

[5] Leaute, T., Ottens, B., Szymanek, R.: *FRODO 2.0: An open-source framework for distributed constraint optimization*. In Proceedings of the IJCAI'09 Distributed Constraint Reasoning Workshop (DCR'09). pp. 160–164. Pasadena, California, USA, 2009. Available: http://liawww.ep.ch/frodo/.

[6] Lytinen, S. L. and Railsback, S. F. *The evolution of agent-based simulation platforms: A review of NetLogo 5.0 and ReLogo*. In Proceedings of the Fourth International Symposium on Agent-Based Modeling and Simulation, Vienna, 2012.

[7] A. Meisels, *Distributed Search by Constrained Agents: algorithms, performance, communication*, Springer Verlag, London, 2008.

[8] Modi, P., Shen, W.-M., Tambe, M., Yokoo, M., *ADOPT: Asynchronous distributed constraint optimization with quality guarantees*, A.I., 2005, 161 (1-2), pp. 149-180.

[9] Monier, P., Piechowiak, S. et Mandiau, R. *A complete algorithm for DisCSP: Distributed Backtracking with Sessions (DBS)*. In Second International Workshop on: Optimisation in Multi-Agent Systems (OptMas), Eigth Joint Conference on Autonomous and Multi-Agent Systems (AAMAS 2009), Budapest, Hungary, pp 3946.

[10] Muscalagiu I, Popa HE, Vidal J. *Large Scale Multi-Agent-Based Simulation using NetLogo for implementation and evaluation of the distributed constraints. In proceedings of IJCAI DCR 2013 (23rd International Joint Conference on Artificial Intelligence - Workshop on Distributed Constraint Reasoning*, 2013, Beijing, August, pp. 60 - 74.

[11] Muscalagiu I, Popa HE, Vidal J. *Enhancing DisCSP-Netlogo from simulation to real-execution of agents in distributed constraints. In proceedings of 18th Annual International Conference on Knowledge-Based and Intelligent Information and Engineering Systems (KES 2014)*, Procedia Computer Science 35 (2014) 261 270.

[12] A. Petcu. *A Class of Algorithms for Distributed Constraint Optimization*. PhD. Thesis No. 3942, Swiss Federal Institute of Technology (EPFL), Lausanne, 2007.

[13] Sultanik, E.A., Lass, R.N., Regli,W.C. *Dcopolis: a framework for simulating and deploying distributed constraint reasoning algorithms*. AAMAS Demos, page 1667-1668. IFAAMAS, (2008).

[14] Silaghi M.C., D. Sam-Haroud, B. Faltings. *Asynchronous Search with Aggregations*. In Proceedings AAAI'00, 917-922.

[15] S. Tisue and U. Wilensky. *Netlogo: Design and implementation of a multi-agent modeling environment*. In Proceedings of the Agent 2004 Conference, 2004, pp 7 - 9.

[16] M. Yokoo, E. H. Durfee, T. Ishida, K. Kuwabara. *The distributed constraint satisfaction problem: formalization and algorithms*. IEEE Transactions on Knowledge and Data Engineering 10(5), page. 673-685, 1998.

[17] U. Wilensky. *NetLogo itself: NetLogo*. Available: http://ccl.northwestern.edu/netlogo/. Center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, 1999.

[18] Wilensky U, Stroup W. *HubNet*. Center for Connected Learning and Computer-Based Modeling, Northwestern University: Evanston, IL USA, ¡http://ccl.northwestern.edu/ps/¿, 1999.

[19] M. Wahbi, R. Ezzahir, C. Bessiere, El Houssine Bouyakhf. *DisChoco 2: A Platform for Distributed Constraint Reasoning*. Proceedings of the IJCAI11 Workshop on Distributed Constraint Reasoning (DCR11), pages 112–121, Barcelona, 2011.

[20] J. Vidal. *Fundamentals of Multiagent Systems with NetLogo Examples*. Available: http://multiagent.com/p/fundamentals-of-multiagent-systems.html.

[21] *MAS NetLogo Models-a*. Available: http://discsp-netlogo.fih.upt.ro/.

[22] *MAS NetLogo Models-b*. Available:http://jmvidal.cse.sc.edu/netlogomas/.

[23] *InfraGRID Cluster*. Available:http://hpc.uvt.ro/infrastructure/infragrid/