Implementation and evaluation methodology for the asynchronous search techniques in DisCSP-NetLogo

IONEL MUSCALAGIU, MANUELA PANOIU, DIANA MARIA MUSCALAGIU, CAIUS PANOIU
The "Politehnica" University of Timisoara
The Faculty of Engineering of Hunedoara
Revolutiei, 5, Hunedoara
ROMANIA
mionel@fih.upt.ro, m.panoiu@fih.upt.ro, dianamariamuscalagiu@yahoo.com, c.panoiu@fih.upt.ro

Abstract: The implementation and evaluation of asynchronous search techniques can be done in any programming language allowing a distributed programming. Nevertheless, for the study of such techniques and for their evaluation, it is easier and more efficient to implement the techniques under certain distributed environments, which offer various facilities, such as NetLogo and are open-source. This article proposes a solution of evaluation for the asynchronous search techniques in NetLogo. This model will allow the use of the NetLogo environment as a basic simulator for the evaluation of asynchronous search techniques. Starting from the proposed evaluation model, we obtain one multi-agent system which can be used for implementing and evaluating the asynchronous search techniques, that can run on a single computer. In this paper there is presented a methodology of implementation and evaluation for the asynchronous search techniques in NetLogo, using the proposed evaluation model.

Key–Words: constraints programming, distributed problems, asynchronous searching techniques, multi-agent systems.

1 Introduction

Constraint programming is a programming approach used to describe and solve large classes of problems such as searching, combinatorial and planning problems. Lately, the AI community has shown increasing interest in the distributed problems that are solvable through modeling by constraints and agents. The idea of sharing various parts of the problem among agents that act independently and collaborate in order to find a solution by using messages proves itself useful. It has also lead to the formalized problem known as the Distributed Constraint Satisfaction Problem (DisCSP) [5].

There exist complete asynchronous searching techniques for solving the DCSP, such as the ABT (Asynchronous Backtracking) and DisDB (Distributed Dynamic Backtracking) [1, 5]. There is also the AWCS (Asynchronous Weak-Commitment Search) [5] algorithm which records all the nogood values. The ABT algorithm has also been generalized by presenting a unifying framework, called ABT kernel [1]. From this kernel two major techniques ABT and DisDB can be obtained.

The asynchronous search techniques, existent for the DisCSP modeling, are within the framework of distributed programming. The agents can be processes residing on a single computer or on several computers, distributed within a network or Internet. 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 can be done in any programming language allowing a distributed programming, such as Java, C, C++. 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, which offers such facilities (NetLogo [6], [9]).

NetLogo, is a programmable modeling environment, which can be used for simulating certain natural and social phenomena [6]. 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. NetLogo is the next generation in a series of modelling languages with agents that began with StarLogo [6]. It is a environment written entirely in Java, therefore it can be installed and activated on most of the important platforms.

The aim of this article is to introduce an model of evaluation for the asynchronous search techniques in NetLogo, by extending the model in [3] and [4], model calling DisCSP-NetLogo. This model can be used in the study of agents behavior in several situ-

ations, like the priority order of the agents, the synchronous and asynchronous case, leading, therefore, to identifying possible enhancements of the performances of asynchronous search techniques. We extend the model in [3] and [4] with support for the evaluation of the performances of asynchronous search techniques and multi variable per agent. In this paper there is presented a methodology of implementation and evaluation for the asynchronous search techniques in NetLogo, using the proposed evaluation model.

Starting from the proposed implementation and evaluation model, we obtain one multi-agent system which can be used for implementing and evaluating the asynchronous search techniques. Implementation examples for the ABT family and the AWCS family can be found on the website [9].

2 Modeling and implementing of the agents' execution process

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 [3], [4]. This solution, calling DisCSP-NetLogo will be extended with support for the evaluation of the performances of asynchronous search techniques.

This modeling can also be used for evaluation of the asynchronous search techniques, such as those from the AWCS family [5], ABT family [1], DisDB [1]. Implementation examples for these techniques can be found on the NetLogo site ([7]) and in [8], [9].

The modeling of the agents' execution process will be structured on two levels, corresponding to the two stages of implementation. The definition of the way in which asynchronous techniques will be programmed so that the agents will run concurrently and asynchronously will be the internal level of the model. The second level refers to the way of representing the NetLogo application. This is the exterior level. The first aspect will be treated and represented using turtle type objects. The second aspect refers to the way of interacting with the user, the user interface. Regarding that aspect, NetLogo offers patch type objects and various graphical controls.

2.1 The NetLOGO objects

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

A set of agents can be grouped in an *agentset*. An *agentset* can contain any of the objects of the turtles or patches type, but not both. The concept of the set of the agents was introduced in order to apply commands or properties.

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

2.2 Agents' simulation and initialization

First of all, the agents are represented by breed type objects (those are of turtles type objects). Fig. 1 shows the way the agents are defined together with the global data structures owned by the agents.

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

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

The initialization of the agents supposes building the agents and initialization of the necessary data structures for the agents' operation. For initialization there is proposed an initialization procedure for each agent, procedure presented in [3], [4].

2.3 Representation and manipulation of the messages

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

- set message (list ”ok” agent value agent-costs) - messages of the ok or info type;
- set message (list ”nogood” agent current-view agent-costs) - messages of the nogood or back type;
- set message (list ”addl” agent1 agent2 agent-costs);
- set message (list ”removel” agent1 agent2 agent-costs);

We can remark the fact that the definition of a message contains the message type, the agent that has transmitted and the informations corresponding to the type of message.

The agents'communication is done according to the communication model introduced in [5]. 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 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. In the proposed model in this article, that structure will be called *message-queue*. This structure proper to 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 update for global manipulation of the message channels. It will also have a role in detecting the termination of the asynchronous search techniques' execution. That update procedure is some kind of a ”main program”, a command center for agents. 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 operation with messages that are transmitted by the agents. The procedure 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. In figures 2 and 3 there are presented the two main procedures for handling and treating messages, update and handle-message. 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).

```
to update
  set no-more-messages true
  ask agents [
    if (not empty? message-queue)[ set no-more-messages false]]
 ;if all queues are empty, the algorithm can be stopped
  if (no-more-messages) [
    if (Solution)
      [WriteSolution]
    else [WriteNoSolution]
    stop]
 ;the procedure for handling messages from the
 ;message queues is called, for each agent.
  ask agents [handle-message]
  set nr-cycles nr-cycles + 1
  ask agents [ ...
     plot messages-received-nogood]
 ;graphical representations can be made after various parameters
  . . .
end
```

Figure 2: The NetLogo update-1 procedure

In the update procedure from fig. 2, can be noticed the call ”ask agents ....”, which allows the execution of the computations for each agent, but with a synchronization after each run of the agents. In fig. 3, where the procedure for message handling is presented, it can be observed the fact that each agent extracts a message at a time from the message queue, identifies the message type and calls the appropriate procedure for handling that type of message.

```
;the message from the queue is handled and its type is
identified(ok, nogood, etc.) calling the handling procedure
to handle-message
  if (empty? message-queue) [stop]
  set msg retrieve-message
  if (first msg = ”ok”)[
    set messages-received-ok messages-received-ok + 1
    handle-ok-message msg
    ;it's called the procedure of handling the messages ok]
  if (first msg = ”nogood”)[
    handle-nogood-message msg
    ;it's called the procedure of handling the messages nogood
    set messages-received-nd messages-received-nd + 1]
....
end
```

Figure 3: The message handling-1 procedure

2.4 Termination detection for the asynchronous search techniques

For most of the asynchronous search techniques, 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 [3] two detection solutions for the execution termination of the asynchronous search techniques are presented.

The first solution of termination detection is based on some of the facilities of the NetLogo environment: the ask 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 realized by this central agent. For this purpose, the building of an "update" procedure is proposed for global handling the message channels. In such a procedure, that must run continuously (until the message queues are emptied) the observer agent verifies any break in message transmission has been detected. These elements will lead to a variant of implementation in which the synchronizing of the agents' execution is done. This method allows obtaining a multi-agent system with synchronization of the agents' execution.

The second method allows us to obtain implementations with a completely asynchronous behavior of the agents [3]. In this case, each agent executes asynchronously and concurrently its computations, having no synchronization. This situation is solved by introducing indicators, which retain at each moment the status of the communication channels for each agent.

The second method of detection starts from renouncing the role of the "observer" to detect the apparition of a break in message transmission and execution of the update procedure by each agent. The detection will be made by the first agent that detects its apparition. To each agent will be attached an indicator detecting its local status. The detecting of the algorithm's termination is done using these indicators that will store at any time the status of the communication channels for each agent. If, at a given time, all the indicators are true, we can stop the algorithm. In fig. 4(a) the code of the update procedure is presented.

The use of the second method of detection of termination requires the adaptation of the message manipulation routine, obtaining the routine from fig. 4(b). The second solution allows the building of another type of evaluation system in NetLogo, with agents, without synchronization for the agents' execution. This system allows studying the agents' behavior in conditions closer to the real ones.

```
to update
  ; the procedure for handling messages
  ; from the message queues is called
  handle-message
  ;the first agent that notices that all
  ;queues are empty will stop the algorithm
  if (sum from agents [gt])=num-agents)
    [ set no-more-messages true ]
  if (no-more-messages) [
    if (Solution)
      [ WriteSolution ]
    else [ WriteNoSolution ]
    stop]
end
```

(a) The update-2 procedure

```
to handle-message
  if (empty?message-queue) [ set gt 1
    stop]
  set msg retrieve-message
  set gt 0
  if (first msg = "ok")[
    set messages-ok messages-ok + 1
    handle-ok-message msg ]
  if (first msg = "nogood")[
    handle-nogood-message msg ]
....
end
```

(b) The handle-message-2 procedure

Figure 4: The new procedure (version 2)

Starting from the modeling of the process of the agents' execution proposed in the previous paragraph, applying a method for the detection of the termination of this process, we can obtain two multi-agent systems that can be used to evaluate the asynchronous search techniques [3].

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 evaluation of the asynchronous search techniques depends on at least two factors: the types of problems used at the evaluation and the units of measurement used. Each problem can be used with a certain efficiency of a certain technique, depending on the problem's difficulty. For the CSP modeling there were used some types of classic problems: the n queens problem, the m coloring of a graph problem or the SAT problem. These problems were taken over for the analysis of the DisCSP techniques in the distributed formulation in which the variables where

taken over by agents. For these problems there are a few parameters that define them. The most important are the dimension of the problem and the density of the constraints graph associated to the DisCSP problem.

To evaluate these techniques there are many ways of measuring that ensure a certain independence from the programming languages used to implement them. These measurement units allow the evaluation of asynchronous search techniques according to many criteria, such as local and global effort of the agents, network loading due to the message exchange.

3.1 Types of problems used in evaluation

There a few types of problems about the evaluation in the DisCSP literature:

- the distributed problem of the n queens, characterized by the number of queens (constant density for the constraints graph equal to n*(n-1)/2).
- the distributed problem of the m-coloring of a randomly generated graph, characterized by the number of nodes/agents, k=3 colors and the m-number of connections between the nodes/agents. There are defined two types of graphs: graphs with few connections (known as sparse problems, having m=n x 2 connections) and graphs with a special number of connections (known as dense problems, m=n x 2.7).
- The randomly generated (binary) CSPs are characterised 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. That is, p1 may be thought of as the density of the constraint graph, and p2 as the tightness of constraints.

There should be mentioned that the problem of the coloring of a graph and the randomly generated binary are the most suitable for the evaluation, because they allow different densities for the constraints graph and they have many direct applications in real practice. Therefore, a correct evaluation supposes the selection of a varied class of problems, the more dimensions, the more sets of data chosen randomly, or the choosing of sets of data which allow varied densities for the constraints graph.

Unfortunately, each asynchronous running of the same problem can give different results. This problem is solved by multiple runnings of the same problem, for the same set of data, by selecting the arithmetic average and the dispersion.

For the types of problems chosen for the evaluation of the asynchronous search techniques there will

```
breeds [nodes]
;edges are represented in a list of lists:a 2D array indexed by "who".
globals [edges domain no-more-messages done nr-cycles]
;domain is the list of allowed colors;
nodes-own [message-queue current-view MyValue nogoods
messages-received-ok,..., the-neighbors, the-links]
;;the-neighbors is a list of the initial neighbors nodes
;message-queue contains the input messages.
;current-view is a list indexed by agent number [c0 c1...]
;nogoods is the list of inconsistent positions [0 1 1 0 ... ]
where 0 is a good position and 1 is inconsistent.
;messages-received-ok count the number of ok messages
received by an agent.
```

Figure 5: NetLogo agents in the case of the graph coloring problem

```
breeds [nodes]
;nodes = agents
globals [edges domain no-more-messages done nr-cycles]
;domain is the list of allowed colors;
nodes-own [message-queue current-view MyValue nogoods
messages-received-ok,..., the-neighbors, the-links, Forbidden-Pairs]
;the-neighbors is a list of the initial neighbors nodes
;message-queue contains the input messages.
;current-view is a list indexed by agent number [c0 c1...]
;nogoods is the list of inconsistent positions [0 1 1 0 ... ]
where 0 is a good position and 1 is inconsistent.
;messages-received-ok count the number of ok messages
received by an agent.
;Forbidden-Pairs is a list of conflicting pairs of values
```

Figure 6: NetLogo agents in the case of the randomly generated (binary) CSPs problem

be defined agents using constructions of the breeds type. That thing will allow programming each DCSP agent according to the chosen asynchronous search techniques. Defining the agents in the case of the two classic problems used for evaluation (the problem of graph coloring and the randomly generated (binary) CSPs) is presented in figures 5 and 6. When the agents are defined, the proprietary data structures are also defined.

3.2 Costs due to the communication

The first criterion is that of the costs due to the communication of information between different parts of the algorithm. The asynchronous behavior that we've met in the asynchronous searching techniques is influencing in a substantial way the communication costs. The asynchronous algorithms are characterized by the usage of messages from the agents during the solution seeking time. The monitoring of the received messages allows the evaluation of global charging of the network. It's important to analyze for distributed en-

vironments that use Internet or Ethernet networks.

The model presented within this paper allows the monitoring of the various types of messages used by the asynchronous search techniques. This can be done by using a few global variables that are attached to agents. For instance, for the model presented, there can be used a variable proprietary to each agent (messages-received), variable that must be incremented when generating and sending a message. This variable is incremented in the routine of manipulation of the messages of the type handle message (fig. 2 and fig. 3(b)).

3.3 Time costs

The time complexity is given by the time that is necessary for calculating, and it is expressed in the terms of the message with the longest treating time that the computation involves. This time complexity is comparable to the time complexity of the algorithm for the sequential classic case, but this is not really a time measurement.

The asynchronous search techniques are evaluated by certain authors [5], using for the time complexity, as a measurement unit, the cycle. A cycle consists in 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 measurement unit assures a certain independence to the local conditions and the ones existing in the distributed environment, impossible to be influenced by the delays that occur. This is a very good measurement unit for the environments that simulate a real distributed environment. For the model featured in this study, the number of cycles can be determined by using a global NetLogo variable (without being proprietary to the agents). This variable will be incremented in the *update* routine. In fact, the number of calls for the Update procedure represents the number of cycles executed by the agents to obtain the solution, or the non-existence of the solution. The number of cycles can be determined only in the case of the multi-agent system with the agents' execution synchronized (fig. 3).

The time complexity can be also evaluated by using the total number of constraints verified by each agent. There 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 agent (nr-constraintc). A counter like that is incremented within the routines check-agent-view, for verifying the consistency of agent's value.

1: Each agent initializes the counter variables CounterList with 0.
Also, MaxCounter is initialized with 0.
2: When an agent sends a message it includes in the message
the value of it's MaxCounter.
3: When an agent receives a outdated nogood value from a
Sender agent, the counter from the CounterList list is updated.
Replace-item Sender CounterList **with** item Sender CounterList+1
4:**if** an agent receives a message with a counter SenderMx
then
Set MaxCounter = max { CounterList }
if MaxCounter < SenderMx **then**
Set MaxCounter = SenderMx
end if
end if

Figure 7: Determining the maximum value received by each agent

The counting of the concurrent constraints induced in [2] is a more complicated problem. 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 through the messages. Each agent, when receiving a message that contains a value SenderC-Cost, will update its own monitor AgentC-Cost with the new value according to the algorithm in [2].

Performance improvement for the asynchronous search techniques supposes knowing some informations by all the agents. It is the case of the nogood processor techniques or of the limiting of the number of transmitted messages by each agent. These informations should be known by each agent at a given time. The presented model allows two solutions. A first solution consists in using some global variables, nonproprietary, accessible to all the agents. Such a solution corresponds to the existence of a central agent that stores these informations. But, such a solution isn't close to the practical situation in which the agents cannot communicate with such a central agent or can communicate, but the costs for obtaining a solution increase. The second solution is based on the idea of the algorithm for determining the number of concurrent constraints. Starting from that idea there is proposed in figure 7 an algorithm for determining the maximum/minimum values known by each agent, using only the messages transmitted by the agents.

Each agent uses a list of counters that store the values received from each agent (for example the nogoods received, the number of exchanged values, etc). Then, each agent computes in a proprietary variable the maximum/the minimum from its data structure. When receiving a message, it updates that maximum/minimum with the value transmitted by another agent. Because the agents have connections

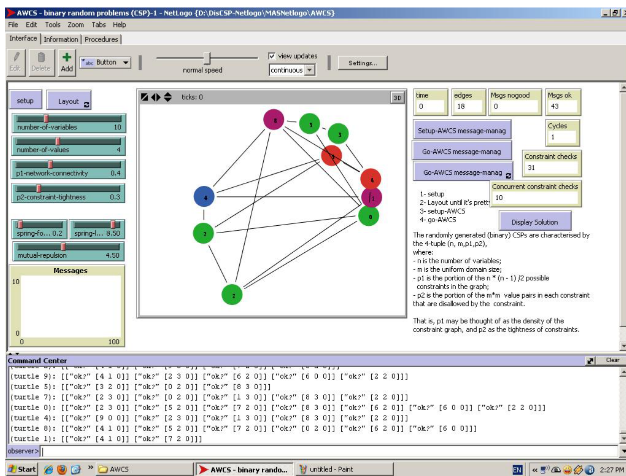

Figure 8: NetLogo implementation and evaluation for AWCS technique

based the constraints, they will receive the maximums/minimums from the neighbor agents, and those from other neighbors, and so on.

Application of the model presented previously allows the implementation and evaluation of any asynchronous search technique. In figure 3.3 there is captured an implementation for the AWCS technique that uses the model presented. Implementation examples for the ABT family and the AWCS family can be downloaded from the website [9].

4 Implementation and evaluation methodology for the asynchronous techniques

In this paragraph there is presented a methodology of implementation for the asynchronous search techniques in NetLogo, using the model presented in the proposed evaluation model. That methodology supposes the identification of the application's objects, building the agents and of the working surface for the application. There are also built the communication channels between agents, routines for message handling and the main program of the application.

The methodology contains more elements specific to NetLogo necessary for finalizing the implementation and evaluation of the asynchronous search techniques. Any implementation based on the presented model, will require the following of the next steps.

P1. Defining the DisCSP application's objects. Starting from the type of problem that is implemented, the objects of the DCSP application will be defined. In figure 9 there is presented a solution of agents modelling and also for the working surface of the application. As in the modelling examples there are proposed breeds [queens] (for modelling the agents associated to the queens from the problem of the n queens) or breeds [vertices] (for modeling the agents associated to each node from the problem of graph coloring).

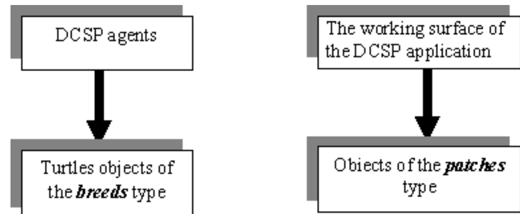

Figure 9: Defining the DisCSP application's objects.

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

P2. Messages handling. Any agent keeps its working context at least as two proprietary structures: current-view and its nogood list. That context is used to take decisions, inclusively for building messages. For the proposed model, the data structures that store the working context of each agent can be simulated with lists. A representation solution is presented in figure 10 (a).

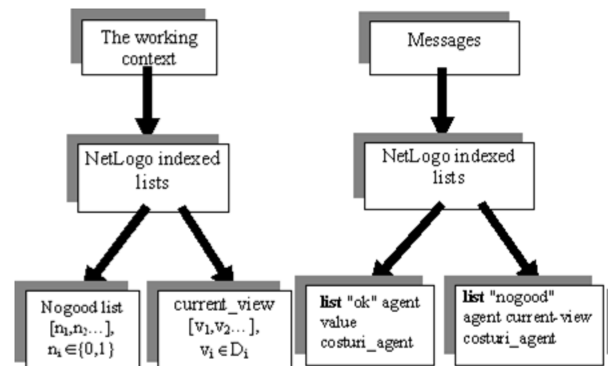

Figure 10: Messages handling

Message handling supposes first of all message representation. That can be realized using indexed lists. For representation of complex messages, that contain a lot of information, Netlogo allows using lists of lists (the elements of the list are also lists). In figure 10(b) there is presented the way of representation of the main messages found at the asynchronous techniques.

Simulation of message queues for each agent can be done using Netlogo lists, for which are defined routines of handling corresponding to FIFO principles. These structures keep the messages received by each agent.

P3. Application initialization. ”The main program” for the application.

The initialization of the application supposes the building of agents and of the working surface for them. When the agents are built the required initializations are also done. Usually, there are initialized the working context of the agent (current-view), the message queues, the variables that count the effort carried out by the agent. In figure 11 there are presented the two routines of the application initialization and of the agents' initialization.

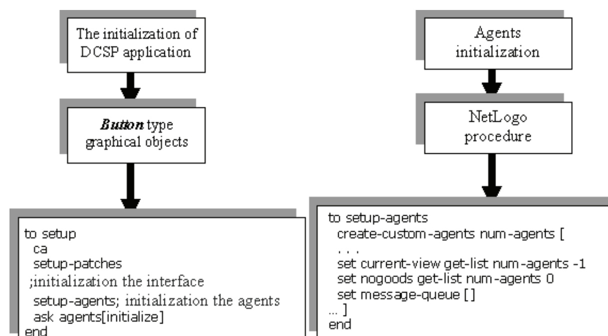

Figure 11: Application initialization

The working surface of the application should contain NetLogo objects through whom the parameters of each problem could be controlled: the number of agents, the density of the constraints graph, the number of colors. For that there can be used Netlogo objects of the slider type. These objects allow the definition and monitoring of each problem parameters.

For the application running there 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 (which in NetLogo stops the execution of an agent). The solution presented in figure 12 is based on the utilization of the ask command. That NetLogo command executes a synchronization of each agent's execution.

Another important observation is tied to attaching the graphical button to the observer. The use of this solution allows obtaining a solution of implementation with synchronization of the agents' execution. In that case, the observer will be the one that will initiate the stoping of the DisCSP application execution. In figure 12 the update procedure is attached and handled by the observer. These elements lead to the multi-agent system with synchronization of the agents' execution.

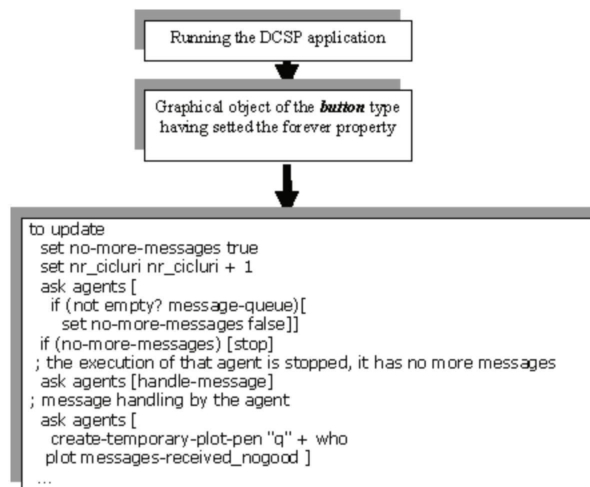

Figure 12: The procedure for running the DisCSP application

If it is desired to obtain a system with asynchronous operation, there will be used the second method of detection, which supposes another update routine. That new update-2 routine will be attached to a graphical object of the button type which is attached and handled by the turtle type agents (the simulated agents were of the breed type, which is a particular case of the turtle type).

P4. Monitoring of the evaluation parameters.

The model presented in this chapter allows storing the costs for obtaining the solution. That thing can be done using some variables attached to the agents. For counting the flow of messages there can be used a variable proprietary to each agent (messages-received-nogood and messages-received-ok), variable that needs to be incremented in the moment of receiving a message. That variable is incremented in the routine of message manipulation handle-message. For measuring the work effort carried out by the agents there can also be used two variables nr-constraintc and concurent-ccks. Those variables store the costs necessary for each agent. Thus, those costs should be measured. An example is presented in figure 13.

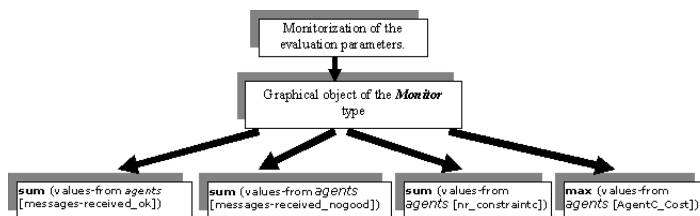

Figure 13: Monitoring of the evaluation parameters for the two multi-agent systems

Application of the methodology presented previously allows the implementation and evaluation of any asynchronous search technique.

Starting from the modelling of the process of the agents' execution proposed in the previous paragraph, applying a methodology presented previously, we can obtain two multi-agent systems that can be used to implement and evaluate the techniques of asynchronous search [3]. The architectures of the two distributed systems that can be used for the asynchronous search techniques are presented in figures 14 and 15.

The two systems are characterized by some common elements. First of all the agents simulation is made in the same way, each agent being simulated through the use of breed type objects. Second, we can attach a general message treating routine to the agents (in figures 4(a) and 4(b) handle-message). Third, the message treating procedures (which, usually differentiate each technique) are implemented in the same way for the two systems.

The main difference between the two systems consists in the detection of the termination of each implemented technique. The first multi-agent system uses the system agent called observator, in the second one the agents are simulated through breed type objects.

The first multi-agent system of implementation and evaluation is obtained by applying the first method of detection of termination for the asynchronous search techniques[3]. This system will be called implementation and evaluation system with synchronization-SIES. The main idea is to use the system observer agent for identifying the break in the message transmission. That thing is done by attaching the update routine to a button attached to the observer.

The second element that differentiates this system is that of using the ask command for executing the handle-message routine of each agent (in the update routine). That solution leads to the synchronization of the agents execution. In figure 14 there is presented this multi-agent system's architecture.

The second multi-agent system is obtained applying the second method of termination detection for the agents' execution at the model of implementation and evaluation proposed previously [3]. That system will be called implementation and evaluation system with asynchronous operation of the agents-SIEAS (figures 15). The multi-agent system is remarked by renouncing the observer and attaching the update routine to each agent of the breed type. That way the execution of the handle-message routine isn't done by the ask command, that being executed for each agent (turtle) by the turtle type button with "forever-button" feature set. Giving up the ask command has the effect of asynchronous run of the agents, each agent processing its messages without waiting for a synchronization with the other agents.

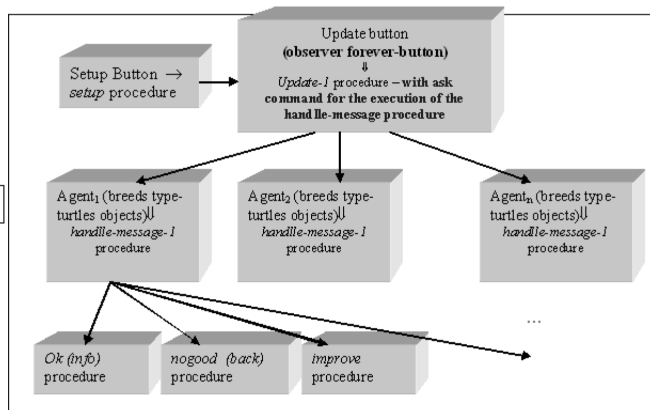

Figure 14: Architecture of a multi-agent system with execution synchronization - SIES

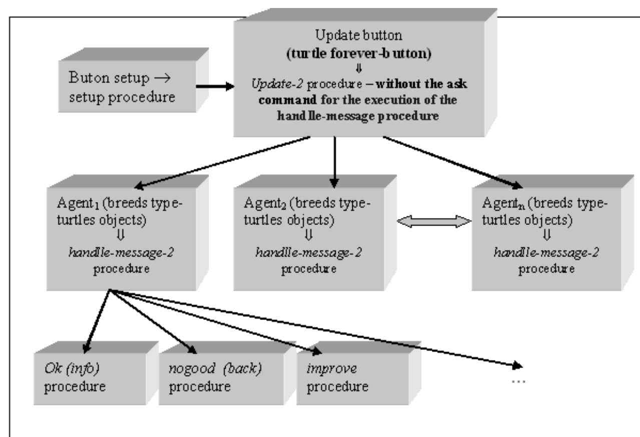

Figure 15: Arhitecture of a multi-agent system with asynchronous agents operation-SIEAS

5 Conclusions

In this article was analyzed the NetLogo environment with the purpose of building a general model of implementation and evaluation for the asynchronous

techniques such as they could use the NetLogo environment as a basic simulator in the study of asynchronous search techniques. The proposed model supposed the identification of NetLogo objects necessary for implementing the asynchronous search technique (agents, messages, message queues, agents ordering) and of the interface of interaction with the user. There were also proposed solutions for counting the costs for obtaining a solution using different measuring units. That thing will allow the evaluation of performances for asynchronous search techniques and eventual improvements for them. We can consider that it is the first NetLogo model that allows the implementation and evaluation of the asynchronous search techniques after many evaluation criteria. The model also allows the studying of the behavior of the agents for various techniques, the studying of the costs for each agent.

Another very important facility is connected to the fact that the NetLogo environment, through the multi-agent systems defined, also allows the simulation of other practical situations, such as the apparition of delays in message transmission. There can be introduced breaks in message processing through handle-message routines and studied the behavior of the agents in the case of apparition of delays in message supplying. There can also be simulated situations in which there also is necessary the control of the channels of communications, for example for message filtering.

Starting from the modelling of the process of the agents' execution proposed in this paper, applying a methodology presented previously, we can obtain two multi-agent systems that can be used to implement and evaluate the techniques of asynchronous search. These multi-agent systems can be used for implementation, evaluation, analysis and identification of improvements for the asynchronous search techniques.

References:

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

[2] A. Meisels, E. Kaplansky, I. Razgon, and R. Zivan. *Comparing performance of distributed constraints processing algorithms*. Notes of the AAMAS'02 workshop on Distributed Constraint Reasoning, pages 86-93, Bologna, Italy, 2002.

[3] Muscalagiu, I., Jiang, H., Popa, H. E. *Implementation and evaluation model for the asynchronous techniques: from a synchronously distributed system to a asynchronous distributed system*. Proceedings of the 8th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, Timisoara, IEEE Computer Society Press, 209–216, 2006.

[4] Muscalagiu, I., Iordan , A., Muscalagiu, D., Panoiu, M. *Implementation and evaluation model with synchronization for the asynchronous search techniques*. Proceedings of the 13th WSEAS International Conference on COMPUTERS(13th WSEAS CSCC), Rhodos Island, Greece,July 23-25, page 211–216, 2009.

[5] 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.

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

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

[8] *MAS NetLogo Models-b*. Available: http://ccl.northwestern.edu/netlogo/models/community.

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