Implementation and evaluation model with synchronization for the asynchronous search techniques

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

Abstract: In this article a general implementation and evaluation model with synchronization and support for message management in Netlogo, for the asynchronous techniques is proposed. This model will allow the use of the NetLogo environment as a basic simulator for the study of asynchronous search techniques. This model can be used in the study of the agents’ behavior in several situations, like the priority order of the agents, the behavior in the synchronous and asynchronous case and, therefore, leading to the identification of possible enhancements in the performances of asynchronous search techniques. Starting from the proposed implementation model, in this article we present two multi-agent systems with synchronization which can be used for the implementation and evaluation of the asynchronous techniques, that can run on a single computer.

Key–Words: constraints programming, distributed problems, asynchronous searching techniques, synchronization

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 (DCSP) [5].

There exist complete asynchronous searching techniques for solving the DCSP, such as the ABT (Asynchronous Backtracking) and DisDB (Distributed Dynamic Backtracking) [2, 5]. There is also the AWCS (Asynchronous Weak-Commitment Search) [5] algorithm which records all the nogood values. This technique allows the agents to modify a wrong decision by a dynamic change of the agent priority order. The ABT algorithm has also been generalized by presenting a unifying framework, called ABT kernel [2]. From this kernel two major techniques ABT and DisDB can be obtained.

The asynchronous search techniques, existent for the DCSP 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. The implementation of any asynchronous search techniques supposes building the agents and the existing constraints, the implementation of the links between the agents and the communication channels between them. The implementation of asynchronous search techniques can be done in any programming language allowing a distributed programming, such as Java, C, C++ or other. Nevertheless, for the study of such techniques, for their analysis and evaluation, it is easier and more efficient to implement the techniques under a certain distributed environment, which offers such facilities (NetLogo [6], [7], [8]).

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. 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 as general as possible model of implementation and evaluation for the asynchronous search techniques in NetLogo, with support for the message synchronization and management, by extending the model in [3]. 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, lead-
ing, therefore, to identifying possible enhancements of the performances of asynchronous search techniques. We extend the model in [3] with support for the message synchronization and management.

Starting from the proposed implementation and evaluation model, we obtain two MAS 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]. This solution will be extended with support for the message synchronization and management.

This modeling can also be used for implementing most of the asynchronous search techniques, such as those from the AWCS family [5], ABT family [2], DisDB [2]. 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 Agents’ simulation and initialization

First of all, the agents are represented by breed type objects (as we saw in the previous paragraph, 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.

2.2 Handling of the messages

Any asynchronous search technique is based on the use of some messages for communicating various information needed for obtaining the solution, exchanged by the agents. The agents’ communication is done according to the communication model introduced in [5].

The communication model existing in the DCSP 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. These data structures are defined in the same time with the definition of the agents. In the proposed implementations 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. These queue type data structures, that have a very important role in the functioning of the asynchronous search techniques, have to respect the FIFO principles.

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 fig. 2 the update procedure is presented.

In the update procedure from fig. 2, can be no-
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]
ask agents [ ...
   plot messages-received-nogood]
;graphical representations can be made
after various parameters (messages, etc.)
....
end

Figure 2: The NetLogo update-1 procedure

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 with a
synchronization after each run of the agents. A variant
for the one above is based on the use of the observer
in handling the message channels. The behavior of
the ask command allows us to build a synchronous
distributed system.

In fig. 3, where the procedure for message han-
dling 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 ap-
propriate procedure for handling that type of message.
The message handling procedures for a particular type
of message can be implemented according to the asyn-
chronous search technique.

2.3 Termination detection for the asyn-
chronous 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). These algo-
rithms are remarkable by the fact that the ending
verification should be initiated by the agents that sus-
pect the occurrence of a break in the sending of mes-
sages.

The finishing of the building of the solution in
NetLogo is based on detecting a break in message transmis-
sion. This situation can be resolved by check-
ing the message queues, queues that need to be empty.
In [3] two detection solutions for the execution termi-
nation of the asynchronous search techniques are pre-
sented.

The second method allows us to obtain imple-
mentations with a completely asynchronous behav-
ior of the agents [3]. In this case, each agent exe-
cutes asynchronously and concurrently its computa-
tions, having no synchronization. This situation is
solved by introducing indicators, which retain at each
moment the status of the communication channels for
each agent. If, at a given time, all the indicators are
true, we can stop the execution of the techniques.

The second method of detection starts from re-
nouncing the role of the "observer" to detect the ap-
parition of a break in message transmission and ex-
ecution of the update procedure by each agent. The
detection will be made by the first agent (of the turtle-
breed type) that detects its apparition. The solution (from NetLogo) is given by the use of some buttons of the type "turtle forever-buttons", buttons that aren’t attached to the observer. Practically, the "ask" command wouldn’t be used to execute the message handling routines. These will be executed by each agent through a key (attached to turtle object) of the "forever-button" type. 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. This thing is done by the first turtle agent that detects that state. In fig. 4(a) the code of the update procedure is presented.

```lisp
(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

```lisp
(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 implement and evaluate the techniques of asynchronous search [3].

3 The transformation of a system into a synchronous one

The model with synchronization is based on NetLogo elements, using the ask command for executing the procedures for treating the agents’ messages. This command does a synchronization of the commands attached to the agents such that the synchronization of the agents’ execution is made automatically.

In this paragraph two solutions of synchronization for the agents’ execution are proposed. These two solutions differ from the initial one on the fact that they are far more general, and don’t depend on the NetLogo language. These two solutions are applied to the multi-agent system with asynchronous agents operation, allowing the obtaining of two multi-agent intermediary systems: with partial and complete synchronization.

3.1 Complete synchronization of the agents’ execution process

The first solution proposed is based on using a common memory zone to which the agents have access and which will store certain information necessary for all the agents. In that common memory zone the value of a global variable Nragents, accessible to all the agents, is stored. Nragents is initialized with the number of agents. Each agent will mark the status of its execution in the variable Nragents. Practically, each agent A_i decrements the Nragents variable with 1 when the message processing routine is executed. Also, when the agent finishes to process the messages from its message queue it increments the value of Nragents with 1. In other words, the variable Nragents allows the identification of the status of the agents’ execution in any moment. At a given moment, Nragents can be equal to the number of agents i.e. all agents have finished to process the messages from the message queue. That could become a moment for synchronization, which can be retained by means of a variable called Syncronization. The first solution allows a complete synchronization of all the agents.

Applying this solution supposes the modification of the two main routines for handling of the messages: the computations routine for each agent (called
update) and the routine for message handling (called handle-message). In figure 5 the new update routine that is performed by each agent is presented. Each agent verifies if the synchronization state is reached. If yes, it runs its own message handling procedure (handle-message). Then, the agent enters the waiting state, until the Nragents variable becomes equal to the number of the agents, in order to pass to a new cycle of execution of its computations.

```plaintext
to update
  if Synchronization
    [ handle-message ]
  if Solution
    [ WriteSolution]
  else [WriteNoSolution]
  stop
] While [Nragents := number-of-agent]
  [ wait
    set Synchronization false
  ] set Synchronization true
end
```

(a) The update-3 procedure

to handle-message
  set Nragents Nragents - 1
  if not empty? message-queue
    [ set gt 0
      set msg retrieve-message
      if (first msg = "ok")
        set messages-ok messages-ok + 1
        handle-ok-message msg
      ... ]
  if ( empty? message-queue )
    [ set gt 1
      stop ]
  set Nragents Nragents + 1
end

(b) The handle-message-3 procedure

Figure 5: The new procedure (version 3)

The two procedures for message handling can be applied to the asynchronous model proposed in paragraph 2 thus obtaining a synchronization of the agents’ execution. Thus, starting from the implementation elements presented in paragraph 2, a third model of implementation and evaluation for the asynchronous techniques, in NetLogo can be obtained.

3.2 Partial synchronization of the agents’ execution

The second solution consists in synchronizing only the neighboring agents. Each agent will wait for its connected neighbors which are placed before him in a lexicographical order, to finish their calculation. This solution allows a partial synchronization of the agents’ execution. It doesn’t require the existence of a central agent [4].

The second solution of partial synchronization is based on the use of a synchronization message. This message is similar to a token that each agent needs to receive in order to carry on with the execution of its computing cycle. For this purpose, each agent uses a second channel of communication for receiving the synchronization messages.

Figure 6(a) shows a new update routine that is performed by each agent, for the second solution. Each agent verifies if the synchronization state is reached, if it has received the synchronization messages from the neighboring agents. In figure 6 (b) the new procedure for message handling is presented. The difference with respect to the first solution is that, after message processing, the synchronization message is sent to the neighboring agents [4].

```plaintext
to update
  if Synchronization
    [ handle-message ]
  if Solution
    [ WriteSolution]
  else [WriteNoSolution]
  stop
] While [ hasn’t received the synchronizing message from all its neighbors that are placed after itself in the lexicographical order ]
  [ wait
    set Synchronization false
  ] set Synchronization true
end
```

(a) The update-4 procedure

to handle-message
  if not empty? message-queue
    [ set gt 0
      set msg retrieve-message
      if (first msg = "ok")
        handle-ok-message msg
      ... ]
  if ( empty? message-queue )
    [ set gt 1
      stop ]
  set msg list "sincron" who
  send msg to Neighbors // is sent only to the neighbors that are before him in a lexicographical order
end

(b) The handle-message-4 procedure

Figure 6: The new procedure (partial synchronization)
There are some differences between the two solutions, differences that appear in the two update routines (figures 5(a) and 6(a)) and handle-message (figures 5(b) and 6(b)). For the first solution, the agents enter the waiting state until the variable Nragents becomes equal to the number of agents. In exchange, for the second synchronization solution, an agent waits to receive the synchronization message from all their neighboring agents that are before him, in lexicographical order.

The two message manipulation procedures can be applied in any language chosen for implementation. In particular, these two routines can be applied to the multi-agent system with asynchronous agents operation, thus obtaining a synchronization of the agents’ execution. In this way, starting from the multi-agent system with asynchronous agents operation from [3], one can obtain two other multi-agent systems usable for the implementation and evaluation of the asynchronous techniques.

4 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.

In this article a general implementation and evaluation model with synchronization and support for the message management in NetLogo is proposed, for the asynchronous techniques. The model also allows the study of the behavior of the agents for various techniques, and the study 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. Breaks can be introduced in message processing through handle-message routines and the behavior of the agents in the case of apparition of delays in message supplying can be studied. The simulation of situations in which it is necessary to control the channels of communications, for example for message filtering, is also possible.

These four multi-agent systems can be used for implementation, evaluation, analysis and identification of improvements for the asynchronous search techniques.

References:


