DisChoco 2: A Platform for Distributed Constraint Reasoning

Mohamed Wahbi[1,2], Redouane Ezzahir[3], Christian Bessiere[1] and El Houssine Bouyakhf[2]

[1] LIRMM/CNRS, University Montpellier II, France
[2] LIMIARF/FSR, University Mohammed V Agdal, Morroco
[3] ENSA Agadir, University Ibn Zohr, Morroco
{wahbi,bessiere}@lirmm.fr, red.ezzahir@gmail.com and bouyakhf@fsr.ac.ma

Abstract. Distributed constraint reasoning is a powerful concept to model and solve naturally distributed constraint satisfaction/optimization problems. However, there are very few open-source tools dedicated to solve such problems: DisChoco, DCOPolis and FRODO. A distributed constraint reasoning platform must have some important features: It should be reliable and modular in order to be easy to personalize and extend, be independent of the communication system, allow the simulation of agents on a single virtual machine, make it easy for deployment on a real distributed framework, and allow agents with a local complex problems. This paper presents DisChoco 2.0, a complete redesign of the DisChoco platform that guarantees these features and that can deal both with distributed constraint satisfaction problems and with distributed constraint optimization problems.

1 Introduction

Distributed Constraint Reasoning (DCR) is a framework for solving various problems arising in Distributed Artificial Intelligence. In DCR, a problem is expressed as a Distributed Constraint Network (DCN). A DCN is composed of a group of autonomous agents where each agent has control of some elements of information about the problem, that is, variables and constraints. Each agent own its local constraint network. Variables in different agents are connected by constraints. Agents try to find a local solution (locally consistent assignment) and communicate it with other agents using a DCR protocol to check its consistency against constraints with variables owned by other agents [1,2].

A DCN offers an elegant way for modelling many everyday combinatorial problems that are distributed by nature (e.g., distributed resource allocation [3], distributed meeting scheduling [4], sensor networks [5]). Several algorithms for solving this kind of problems have been developed. Asynchronous Backtracking (ABT [6], ABT-Family [7]), Asynchronous Forward Checking (AFC) [8] and Nogood-based Asynchronous Forward-Checking (AFC-ng) [9] were developed to solve Distributed Constraint Satisfaction Problems (DisCSP). Asynchronous Distributed constraints OPTimization (Adopt) [10], Asynchronous Forward-Bounding (AFB) [11], Asynchronous

Branch-and-Bound (Adopt-BnB) [12] and Dynamic backtracking for distributed constraint optimization (DyBop) [13] were developed to solve Distributed Constraint Optimization Problems (DCOP).

Programming DCR algorithms is a difficult task because the programmer must explicitly juggle between many very different concerns, including centralized programming, parallel programming, asynchronous and concurrent management of distributed structures and others. In addition, there are very few open-source tools for solving DCR problems: DisChoco, DCOPolis [14] and FRODO [15]. Researchers in DCR are concerned with developing new algorithms, and comparing their performance with existing algorithms. Open-source platforms are essential tools to integrate and test new ideas without having the burden to reimplement from scratch an ad-hoc solver. For this reason a DCR 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 multi-agent systems on a single machine;
– make it easy to implement a real distributed framework;
– allow the design of agents with local constraint networks.

In this paper we present DisChoco 2.0,[1] a completely redesigned platform that guarantees the features above. DisChoco 2.0 allows to represent both DisCSPs and DCOPs, as opposed to other platforms. DisChoco 2.0 is not a distributed version of the centralized solver Choco, but it implements a model to solve DCN with local complex problems (i.e., several variables per agent) by using Choco[2] as local solver to each agent. DisChoco 2.0 is an open source Java library which aims at implementing DCR algorithms from an abstract model of agent (already implemented in DisChoco). A single implementation of a DCR algorithm can run as simulation on a single machine, or on a network of machines that are connected via the Internet or via a wireless ad-hoc network, or even on mobile phones compatible with J2ME.

This paper is organized as follows. Section 2 presents the global architecture of DisChoco 2.0. In Section 3, we show how a user can define her problem and solve it using the DisChoco 2.0 platform. Section 4 shows the different benchmarks available in DisChoco and how researchers in the DCR field can use them for evaluating algorithms performance. We conclude the paper in Section 5.

2 Architecture

In order to reduce the time of development and therefore the cost of the design we choose a components approach allowing pre-developed components to be reused. This components approach is based on two principles:

– Each component is developed independently;
– An application is an assemblage of particular components.

[1] `http://www.lirmm.fr/coconut/dischoco/`
[2] `http://choco.emn.fr/`

Figure 1 shows the general structure of DisChoco kernel. It shows a modular architecture with a clear separation between the modules used, which makes the platform easily maintainable and extensible.

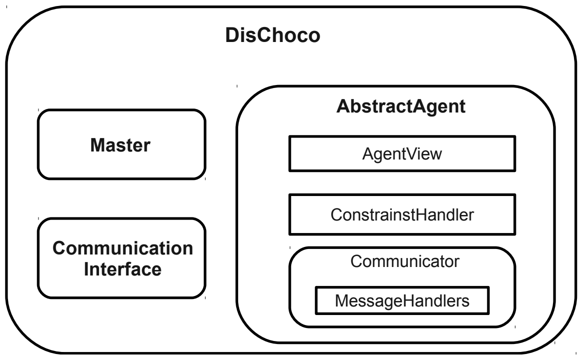


Fig. 1: Architecture of DisChoco kernel

The kernel of DisChoco consists of an abstract model of an agent and several components namely the communicator, messages handlers, constraints handler, the Agent View (AgentView), a Master who controls the global search (i.e., send messages to launch and to stop the search, etc.) and a communication interface.

Communication System

Thanks to independence between the kernel of DisChoco and the communication system that will be used (Figure 2), DisChoco enables both: the simulation on one machine and the full deployment on a real network. This is done independently of the type of network, which can be a traditional wired network or an ad-hoc wireless network.

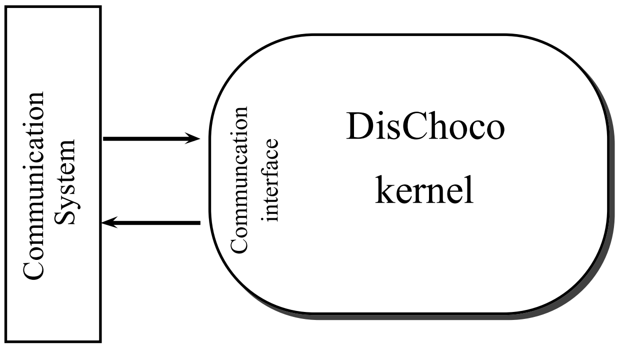


Fig. 2: Independence between the kernel of DisChoco and the communication system

Instead of rewriting a new system of communication between DisChoco agents we adopted the component approach. Thus a communication component pre-developed can be used as a communication system if it satisfies a criterion of tolerance to failure. This allows us to use only the identifiers of agents (IDs) to achieve communication between agents. Thus when agent A_i wants to send a message to the agent A_j, it only attaches its ID (i) and the ID (j) of the recipient. It is the communication interface that will deal with mapping between the IDs and IP addresses of agents (we assume that an agent identifier is unique).

In the case of a simulation on a single Java Virtual Machine agents are simulated by Java threads. Communication among agents is done using an Asynchronous Message Delay Simulator (MailerAMDS) [16,17]. MailerAMDS is a simulator that models the asynchronous delays of messages. Then, agents IDs are sufficient for communication. In the case of a network of Java Virtual Machines, we have used SACI [3] (Simple Agent Communication Infrastructure) as communication system. The validity of this choice has not yet been validated by an in depth analysis. Future work will be devoted to testing a set of communication systems on different types of networks.

Event Management

DisChoco performs constraint propagation via events on variables and events on constraints, as in Choco. These events are generated by changes on variables, and managing them is one of the main tasks of a constraint solver. In a distributed system there are some other events that must be exploited. These events correspond to a reception of a message, changing the state of an agent (wait, idle and stop) or to changes on the AgentView.

The AgentView of a DisChoco agent consists of external variables (copy of other agents variables). Whenever an event occurs on one of these external variables, some external constraints can be awakened and so added to the queue of constraints that will be propagated. Using a queue of constraints to be propagated allows to only process constraints concerned by changes on the AgentView instead of browsing the list of all constraints. To this end, the DisChoco user can use methods offered by the constraints handler (*ConstraintsHandler*).

Detecting the termination of a distributed algorithm is not a trivial task. It strongly depends on statements of agents. To make the implementation of a termination detection algorithm easy, we introduced in the DisChoco platform a mechanism that generates events for changes on the statements of an agent during its execution. A module for detecting termination is implemented under each agent as a listener of events on statements changes. When the agent state changes, the termination detector receives the event, recognizes the type of the new state and executes methods corresponding to termination detection.

The events corresponding to an incoming message are managed in DisChoco in a manner different from the standard method. Each agent has a Boolean object that is set to false as long as the inbox of the agent is empty. When a message has arrived to the inbox, the agent is notified by the change of this Boolean object to true. The agent can

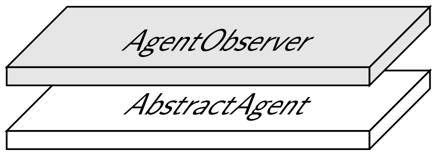
[3] `http://www.lti.pcs.usp.br/saci/`

use methods available in the communicator module to dispatch the received message to its corresponding handler.

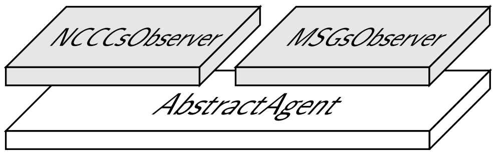
Observers in layers

DisChoco provides a Java interface (*AgentObserver*) that allows the user to track operations of a DCR algorithm during its execution. This interface defines two main functions: *whenSendMessage* and *whenReceivedMessage*. The class *AbstractAgent* provides a list of observers and functions to add one or several observers. Thus, when we want to implement an application using DisChoco, we can use *AgentObserver* to develop a specific observer. This model is shown in Figure 3(a).

When developing new algorithms, an important task is to compare their performance to other existing algorithms. There are several metrics for measuring performance of DCR algorithms: non-concurrent constraint checks ($\#ncccs$ [18]), equivalent non-concurrent constraint checks ($\#encccs$ [19]), number of exchanged messages ($\#msg$ [20]), degree of privacy loss[21], etc. DisChoco simply uses *AgentObserver* to implement these metrics as shown in Figure 3(b). The user can enable metrics when she needs them or disable some or all these metrics. The user can develop her specific metric or her methods for collecting statistics by implementing *AgentObserver*.



(a) *AgentObserver* implemented as Layer



(b) Metrics implemented as observers

Fig. 3: Layer model for observers.

3 Using DisChoco 2.0

Figure 4 presents a definition of a distributed problem named (*Hello DisChoco*) using the Java code. In this problem there are 3 agents $\mathcal{A} = \{A_1, A_2, A_3\}$ where each agent controls exactly one variable. The domain of A_1 and A_2 contains two values $D_1 = D_2 = \{1, 2\}$ and that of A_3 contains one value $D_3 = \{2\}$. There are two constraints of *difference*: the first constraint is between A_1 and A_2 and the second one is between A_2 and A_3. After defining our problem we can configure our solver. Thus, the problem can be solved using a specified implemented protocol (ABT for example).

For DisChoco inputs we choose to use a XML format called *XDisCSP* derived from the famous format XCSP 2.1.[4] Figure 5 shows an example of representation of the problem defined above in the *XDisCSP* format. Each variable has a unique ID, which is the

[4] `http://www.cril.univ-artois.fr/~lecoutre/benchmarks.html`

```java
AbstractMaster master = Protocols.getMaster(Protocols.ABT);
DisProblem disCSP = new DisProblem("Hello DisChoco", master);
SimpleAgent[] agents = new SimpleAgent[3];
IntVar[] variables = new IntVar[3];
// Make agents
agents[0] = (SimpleAgent) disCSP.makeAgent("A1", "");
agents[1] = (SimpleAgent) disCSP.makeAgent("A2", "");
agents[2] = (SimpleAgent) disCSP.makeAgent("A3", "");
// Make one single variable for each agent
variables[0] = agents[0].makeInternalVar(new int[] {1, 2}); // x1
variables[1] = agents[1].makeInternalVar(new int[] {1, 2}); // x2
variables[2] = agents[2].makeInternalVar(new int[] {2}); // x3
// Make two constraints, we must to create external var on each agent
// But each agent must known its constraints
// x1!=x2
agents[0].neqY(agents[0].makeExternalVar(variables[1]));
agents[1].neqY(agents[1].makeExternalVar(variables[0]));
// x2!=x3
agents[1].neqY(agents[1].makeExternalVar(variables[2]));
agents[2].neqY(agents[2].makeExternalVar(variables[1]));
// Make a simulator to resolve the problem
DisCPSolver solver = new DisSolverSimulator(disCSP);
solver.setCentralizedAO(new LexicographicAO());
solver.addNCCCMetric();
solver.addCommunicationMetric();
solver.solve();
System.out.println("Problem : " + disCSP.getProblemName());
System.out.println("Solution of the problem using " + disCSP.master.getClass());
System.out.println("---------------------------------------------------------");
System.out.println(solver.getGlobalSolution());
System.out.println("---------------------------------------------------------");
System.out.println("Statistics :");
System.out.println(solver.getStatistics());
```

Fig. 4: Definition of a distributed problem using Java code.

```xml
<instance>
  <presentation name="Hello DisChoco" model="Simple" maxConstraintArity="2" format="XDisCSP 1.0" />
  <agents nbAgents="3">
    <agent name="A1" id="1" description="Agent 1" />
    <agent name="A2" id="2" description="Agent 2" />
    <agent name="A3" id="3" description="Agent 3" />
  </agents>
  <domains nbDomains="2">
    <domain name="D1" nbValues="2">1 2</domain>
    <domain name="D2" nbValues="1">2</domain>
  </domains>
  <variables nbVariables="3">
    <variable agent="A1" name="X1.0" id="0" domain="D1" description="Variable x_1" />
    <variable agent="A2" name="X2.0" id="0" domain="D1" description="Variable x_2" />
    <variable agent="A3" name="X3.0" id="0" domain="D2" description="Variable x_3" />
  </variables>
  <predicates nbPredicates="1">
    <predicate name="P0">
      <parameters>int x int y</parameters>
      <expression>
        <functional>ne(x,y)</functional>
      </expression>
    </predicate>
  </predicates>
  <constraints nbConstraints="2">
    <constraint name="C1" model="TKC" arity="2" scope="X1.0 X2.0" reference="P0">
      <parameters>X1.0 X2.0</parameters>
    </constraint>
    <constraint name="C2" model="TKC" arity="2" scope="X2.0 X3.0" reference="P0">
      <parameters>X2.0 X3.0</parameters>
    </constraint>
  </constraints>
</instance>
```

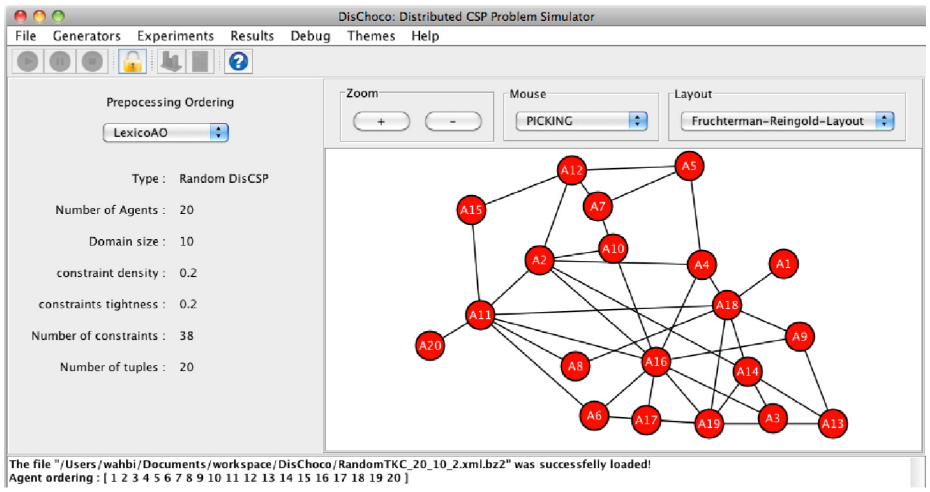
Fig. 5: Definition of the *Hello DisChoco* problem in XDisCSP 1.0 format.

concatenation of the ID of it owner agent and index of the variable in the agent. This is necessary when defining constraints (scope of constraints). For constraints, we used two types of constraints: TKC for Totally Known Constraint and PKC for Partially Known Constraint [21]. Constraints can be defined in extension or as a Boolean function. Different types of constraints are predefined: equal to $eq(x, y)$, different from $ne(x, y)$, greater than or equal $ge(x, y)$, greater than $gt(x, y)$, less than or equal $le(x, y)$, less than $lt(x, y)$,etc.

According to this format we can model DisCSPs and DCOPs. Once a distributed constraint network problem is expressed in the *XDisCSP* format, we can solve it using one of the protocols developed on the platform. The algorithms currently implemented in DisChoco 2.0 are: ABT [6,7], ABT-Hyb [22], ABT-dac [23], AFC [8], AFC-ng [9], DBA [24] and DisFC [21] in the class of DisCSPs with simple agents. In the class of DisCSPs where agents have local complex problems, ABT-cf [25] was implemented. For DCOPs, the algorithms that are implemented in DisChoco 2.0 are: Adopt [10], BnB-Adopt [12] and AFB [11]. For solving a problem, we can use a simple command line:

```
java -cp dischoco.jar dischoco.simulation.Run protocol problem.xml
```

The Graphical User Interface (GUI) of DisChoco allows to visualise the constraint graph. Hence, the user can analyse the structure of the problem to be solved. This also helps to debug the algorithms. An example of the visualisation is shown in Figure 6.



Prepocessing Ordering: LexicoAO

Mouse: PICKING

Layout: Fruchterman-Reingold-Layout

Type : Random DisCSP

Number of Agents : 20

Domain size : 10

constraint density : 0.2

constraints tightness : 0.2

Number of constraints : 38

Number of tuples : 20

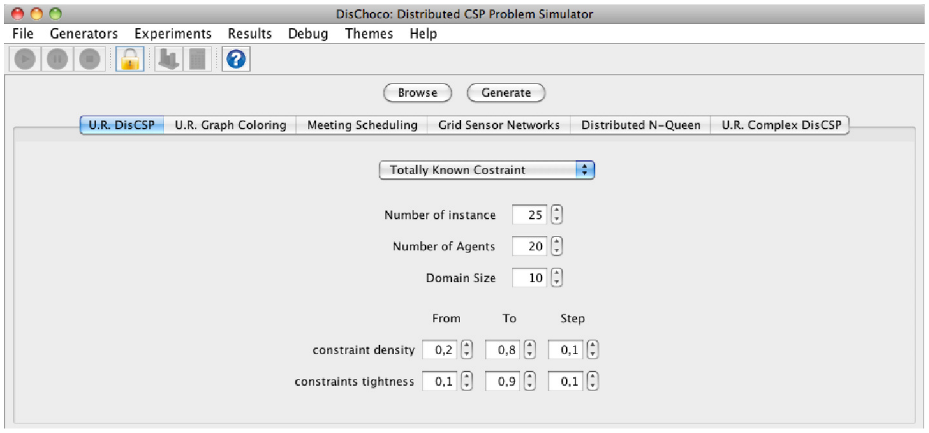
```
The file "/Users/wahbi/Documents/workspace/DisChoco/RandomTKC_20_10_2.xml.bz2" was successfelly loaded!
Agent ordering : [ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 ]
```

Fig. 6: Visualisation of the structure of the distributed constraint graph.

4 Experimentations

In addition to its good properties (reliable and modular), DisChoco provides several other facilities, especially for performing experimentation. The first facility is in the genaration of benchmark problems. DisChoco offers a library of generators for distributed constraint satisfaction/optimization problems (e.g., random binary DisCSPs using model B, random binary DisCSPs with complex local problems, distributed graph coloring, distributed meeting scheduling, sensor networks, distributed N-queens, etc. These generators allow the user to test her algorithms on various types of problems ranging from purely random problems to real world problems.

DisChoco is equipped with a GUI for manipulating all above generators. A screenshot of the GUI of DisChoco shows various generators implemented on DisChoco (Figure 7). Once the instances have been generated, a XML configuration file is created to collect the instances. The generated instances are organized in a specific manner for each kind of problems generator in a directory indicated by the user. The configuration file can also contain details related to the configuration of the communicator and the list of algorithms to be compared. It will be used for launching experiments. After all these configurations have been set, the user can launch the experiments either on the GUI mode or on the command mode.



Totally Known Costraint

Number of instance 25
Number of Agents 20
Domain Size 10

	From	To	Step
constraint density	0,2	0,8	0,1
constraints tightness	0,1	0,9	0,1

Fig. 7: A screenshot of the graphical user interface showing generators in DisChoco

DisChoco is also equipped with a complete manager of results. The user does not have to worry about organizing and plotting results. All this is offered by DisChoco that automatically generates *gnuplot* plots of the requested measures. The user can also handle all results and compare algorithms using the GUI of DisChoco. Figure 8 shows an example of plot generated from experimentations on some algorithms implemented in DisChoco.

5 Conclusion

In this work, we have presented the new version 2.0 of the DisChoco platform for solving DCR problems. This version contains several interesting features: it is reliable and

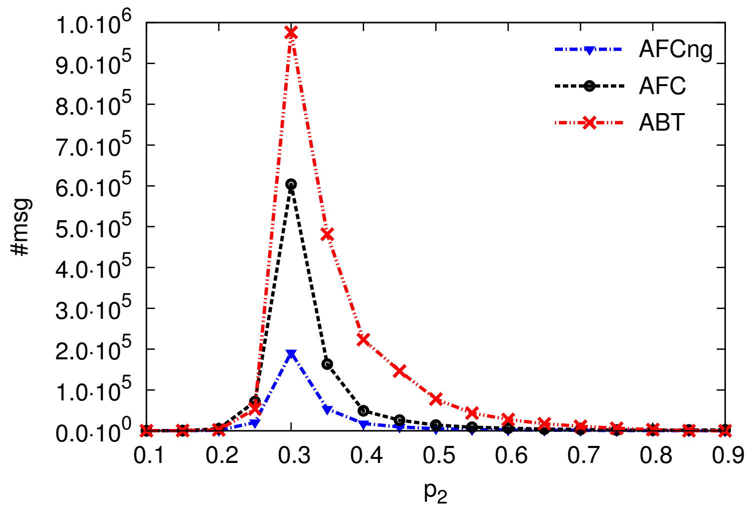


Fig. 8: Total number of exchanged messages on dense graph $\langle$n=20, d=10, $p_1 = 0.7$, $p_2\rangle$.

modular, it is easy to personalize and to extend, it is independent from the communication system and allows a deployment in a real distributed system as well as the simulation on a single Java Virtual Machine. As future work, we aim at enhancing the platform by the implementation of other DCR algorithms and to enrich the graphical user interface to make it easier to use for researchers from the DCR field. Another direction of improvement is to allow DisChoco to support other types of constraints that match as much as possible the needs of real applications. The modularity of DisChoco will allow us to look for other types of system communication. Finally, for a complete validation, it would be interesting to test DisChoco on a real distributed system.

References

1. Yokoo, M., Durfee, E.H., Ishida, T., Kuwabara, K.: Distributed constraint satisfaction problem: Formalization and algorithms. IEEE Transactions on Knowledge and Data Engineering **10** (1998) 673–685
2. Yokoo, M.: Algorithms for distributed constraint satisfaction problems: A review. Autonomous Agents and Multi-Agent Systems **3** (2000) 185–207
3. Petcu, A., Faltings, B.: A value ordering heuristic for distributed resource allocation. In: Proceeding of CSCLP04, Lausanne, Switzerland (2004)
4. Wallace, R.J., Freuder, E.: Constraint-based multi-agent meeting scheduling: effects of agent heterogeneity on performance and privacy loss. In: Proceeding of the 3rd workshop on distributed constrait reasoning, DCR-02, Bologna (2002) 176–182
5. Bejar, R., Domshlak, C., Fernandez, C., Gomes, K., Krishnamachari, B., B.Selman, M.Valls: Sensor networks and distributed CSP: communication, computation and complexity. Artificial Intelligence **161:1-2** (2005) 117–148
6. Yokoo, M., Durfee, E.H., Ishida, T., , Kuwabara., K.: Distributed constraint satisfaction for formalizing distributed problem solving. In: IEEE Intern. Conf. Distrb. Comp. Sys. (1992) 614–621

7. Bessiere, C., Maestre, A., Brito, I., Meseguer, P.: Asynchronous backtracking without adding links: a new member in the ABT family. Artificial Intelligence **161:1-2** (2005) 7–24
8. Meisels, A., Zivan, R.: Asynchronous forward-checking for DisCSPs. Constraints **12** (2007) 131–150
9. Ezzahir, R., Bessiere, C., Wahbi, M., Benelallam, I., Bouyakhf, E.H.: Asynchronous inter-level forward-checking for discsps. In: Proceeding of Principles and Practice of Constraint Programming (CP-09). (2009) 304–318
10. Modi, P.J., Shen, W., Tambe, M., Yokoo, M.: Adopt: asynchronous distributed constraints optimization with quality guarantees. Artificial Intelligence **161:1-2** (2005) 149–180
11. Gershman, A., Meisels, A., Zivan, R.: Asynchronous forward bounding for distributed cops. Journal of Artificial Intelligence Research **34** (2009) 61–88
12. Yeoh, W., Felner, A., Koenig, S.: Bnb-adopt: an asynchronous branch-and-bound dcop algorithm. In: AAMAS'08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems. (2008) 591–598
13. Ezzahir, R., Bessiere, C., Benelallam, I., Bouyakhf, E.H., Belaissaoui, M.: Dynamic backtracking for distributed constraint optimization. In: Proceeding of the 2008 conference on ECAI 2008, Amsterdam, The Netherlands, The Netherlands, IOS Press (2008) 901–902
14. Sultanik, E.A., Lass, R.N., Regli, W.C.: Dcopolis: a framework for simulating and deploying distributed constraint reasoning algorithms. In: AAMAS'08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems. (2008) 1667–1668
15. Léauté, 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), Pasadena, California, USA (2009) 160–164 `http://liawww.epfl.ch/frodo/`.
16. Zivan, R., Meisels, A.: Message delay and discsp search algorithms. Annals of Mathematics and Artificial Intelligence **46** (2006) 415–439
17. Ezzahir, R., Bessiere, C., Belaissaoui, M., Bouyakhf, E.H.: Dischoco: a platform for distributed constraint programming. In: Proceeding of Workshop on DCR of IJCAI-07. (2007) 16–21
18. Meisels, A., Razgon, I., Kaplansky, E., Zivan, R.: Comparing performance of distributed constraints processing algorithms. In: Proceeding of AAMAS-2002 Workshop on Distributed Constraint Reasoning DCR, Bologna (2002) 86–93
19. Chechetka, A., Sycara, K.: No-commitment branch and bound search for distributed constraint optimization. In: Proc. of AAMAS '06, New York, NY, USA, ACM (2006) 1427–1429
20. Lynch, N.A.: Distributed Algorithms. Morgan Kaufmann Series (1997)
21. Brito, I., Meisels, A., Meseguer, P., Zivan, R.: Distributed constraint satisfaction with partially known constraints. Constraints **14** (2009) 199–234
22. Brito, I., Meseguer, P.: Synchronous, asynchronous and hybrid algorithms for DisCSP. In: Workshop on DCR-04, CP-2004, Toronto (2004)
23. Brito, I., Meseguer, P.: Connecting abt with arc consistency. In: CP. (2008) 387–401
24. Yokoo, M., Hirayama, K.: Distributed breakout algorithm for solving distributed constraint satisfaction problems. In Lesser, V., ed.: Proceedings of the First International Conference on Multi–Agent Systems, MIT Press (1995)
25. Ezzahir, R., Bessiere, C., Bouyakhf, E.H., Belaissaoui, M.: Asynchronous backtracking with compilation formulation for handling complex local problems. ICGST International Journal on Artificial Intelligence and Machine Learning, AIML **8** (2008) 45–53