JAIST Repository
https://dspace.jaist.ac.jp/

Title	Distributed Reinforcement of Local Consistency for General Constraint Network An Investigation of Meeting Scheduling Problems
Author(s)	Ahlem, BEN HASSINE
Citation	
Issue Date	2005-09
Type	Thesis or Dissertation
Text version	author
URL	http://hdl.handle.net/10119/819
Rights	
Description	Supervisor:Tu Bao HO, 知識科学研究科, 博士

JAIST
JAPAN
ADVANCED INSTITUTE OF
SCIENCE AND TECHNOLOGY

Japan Advanced Institute of Science and Technology

Distributed Reinforcement of Local Consistency for General Constraint Network
An Investigation of Meeting Scheduling Problems

by

Ahlem BEN HASSINE

submitted to
Japan Advanced Institute of Science and Technology
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

Supervisor: Professor Tu Bao HO

School of Knowledge Science
Japan Advanced Institute of Science and Technology

September 2005

Abstract

Constraint satisfaction problem (CSP) is a powerful formalism to represent and to solve many real-life NP-complete problems such as, planning, resource allocation, meeting scheduling, etc. The great success of this formalism is due essentially to its simplicity in expressing any real-world problem subject to constraints. A CSP is a triplet (X, D, C) composed of a finite set of *n* variables X, each of which is taking values in an associated finite domain D and a set of *e* constraints C between these variables.

Solving a CSP consists in finding one or all-complete assignments of values to variables satisfying all the constraints. However, this task is hard and many efforts were devoted towards enhancing it by reducing the complexity of the original problem. Essentially, the complexity reduction in CSP formalism is achieved by integrating the local consistency property (LC) and its corresponding filtering techniques. Those techniques allow the simplification of the original problem by eliminating values or combination of values that cannot belong to any solution. Many levels of LC have been proposed in the literature, among them enforcing arc-consistency is the most preeminent one because of its low time and space complexities. Most efforts dealing with enforcing AC on any constraint network (CN) are centralized almost always limited to binary CN, i.e., where each constraint involves at most two variables. Non-binary CNs, where constraints involve more than two variables, are often strongly required to deal with hard applications. Nevertheless, there is very few works involving non-binary constraints and they pertain only to the centralized framework.

Recently, with the advent of distributed computing and networking technologies, especially with the omnipresence of naturally distributed real-world problems, the interest in enforcing LC property in naturally distributed manner and for both binary and non-binary CN has largely increased, but such techniques have not been widely studied yet. Moreover, solving real-life applications, mainly meeting scheduling problems, requires also more studies to cope with the new environment requirements.

Our main target is *i*) to find solutions and build a novel generic system to enforce some levels of LC with reasonable cost on any CN and *ii*) to take this system to the real-life through one among the important combinatorial applications, meetings scheduling problems.

Our study on CSP framework and its related research directions including, LC enforcement techniques, and especially ways of solving real applications, mainly meeting scheduling problems (MS) stir up our attention to do more investigations in this framework. Five main contributions of this thesis are the following.

- The integration of LC enforcement techniques in a constraint solver reduces the exponential space, in the number of variables, of the search tree. This clear benefit coupled

with the very few existing research efforts dealing with naturally distributed problems, motivated us to design a new hybrid agent-based method to enforce arc consistency on any CN. This hybrid method involves two main approaches DRAC and G-DRAC, for binary and non-binary constraints, respectively. The termination of the two underlying techniques is guaranteed with equal polynomial time complexity as the best existing distributed technique for DRAC and the best centralized technique for G-DRAC[1] down to the number of variable. As for the spatial complexity, both techniques DRAC and G-DRAC save as much space as possible compared to existing ones. The empirical study of DRAC and G-DRAC shows their efficiency for especially hard problems (Chapter5.).

- Enforcing only arc consistency for some hard CN is fruitless. The main reason is that the problem could be initially AC, thus the filtering process will not prune any values, or prune only few inconsistent values. Achieving higher level of LC could be worthwhile. The main deal here is to find a good compromise between the level to enforce and its cost. Note that no distributed techniques for achieving higher level than AC exist in the literature. We designed an agent-based technique, that we called DRAC^{++} to enforce restricted path consistency, a stronger level than AC with reasonable cost. Moreover, a new heuristic is described in this work to decrease the practical complexity of DRAC^{++}. The experimental results exhibit the efficiency of this new approach towards over-constrained problems (Chapter6.).

- Taking our research results to a real life combinatorial application was our main motivation for the next contribution. Therefore, we choose to evaluate the performance of DRAC on a real decision-making problem: Meeting Scheduling problem (MS). This problem is one the traditional real world problems that continues to fascinate many researchers. This problem embodies a decision-making process affecting several users, in which it is necessary to decide when and where one or more meeting(s) should be scheduled according to several restrictions related to users, meetings, environment, etc. Evidently, solving MS problems, is always time consuming, iterative and also tedious. Despite the continuous efforts of many researchers, this problem needs more investigations to arise many daily encountered difficulties due to the incremental environment requirements. However, DRAC is a filtering technique; it cannot solve any problem but only reduce it without loss of solutions. We came up in this thesis with another novel, agent-based, complete, and deterministic approach (that we called MSS for meeting scheduling solver) to reduce and solve any MS with predictable structure. The proposed underlying protocol is based on a *selfish* welfare to reach the best solution with polynomial cost. The experimental comparisons performed using a typical MS solver show the high performance and scalability of MSS, at least for the used data

[1]There is not any technique to enforce arc consistency on non-binary CN in a naturally distributed manner.

set (Chapter7.).

- The previous work requires a total knowledge about all the meetings in advance. However, for some organizations knowing all meetings beforehand might be quiet difficult rather impossible. This motivated us to tackle a new direction for MS problems, problems with unpredictable structure. Therefore, another more sophisticated solution to solve any dynamic MS problem is described in this thesis. The new technique, that we called MSRAC, is an incremental approach, able to cope with any system alterations, and consequently process any meetings' conflict using three different heuristics. An empirical study highlights the benefit of using the *metropolis criterion* in case of conflict against other heuristics. Moreover, the main goal in MSRAC is to maximize the global system welfare, defined by the optimal solution, while scheduling dynamic meetings (Chapter8.).

- Finally, our last contribution in this thesis is a novel, constraint-based asynchronous search approach (that we called DisAS for distributed asynchronous search). This work is able to tackle directly any constraint network (with non-binary constraints). The proposed approach is based in a part on a *lazy* version of the G-DRAC approach, and without adding any new links and without recording any nogoods as for the existing techniques in the literature. The idea behind using a *lazy* version of G-DRAC is to save as many as possible fruitless backtracking and consequently to enhance the efficiency of the solving process. Furthermore, a new generic distributed method to compute a static constraints ordering were also proposed with DisAS in order to establish an optimal ordering between agents, in which we save as many links as possible leading hopefully to decrease the set of exchanged messages and make it of a great practical use. The designed technique is generic and can be used to solve any naturally distributed real application (Chapter9.).

Keywords: Constraint satisfaction problem (CSP), Distributed CSP, Valued CSP, Local consistency, filtering techniques, Arc consistency, Multi-agent systems, Meeting scheduling problems, Asynchronous backtracking techniques.

Acknowledgments

This work has been supported for one year (2004-2005) by the Japanese Foundation for C&C Promotion Grant for non-Japanese Researcher.

First, I am grateful to Prof. Takuya Katayama of the Japan Advanced Institute of Science and Technology (JAIST) for offering me the opportunity to continue my study in JAIST by sponsoring my research under his COE project, for his encouragements, and for his continuous support.

I would like also express my sincere and deepest thanks to my supervisor Prof. Tu Bao Ho, first for accepting me in his laboratory, for his guidance and for his helpful and valuable comments and suggestions.

I would like to thank my supervisor during the first six months of my PhD degree, Prof. Takayuki Ito for accepting me in his laboratory, guiding me in the first steps of my research, for his kind support, and especially his continuous encouragements to improve my work.

I feel lucky to have close work with my two supervisors, I have benefit from their knowledge, experience, and continuous advises.

My deep thanks also goes to Associate Prof. Xavier Défago, at JAIST, with whom I collaborated during my sub-theme and who offered me, advices and kind guidance during our collaborative work.

I am also very grateful to Associate Prof. Adel El cherif, Associate Professor at the University of Qatar, for encouraging and helping me to come to JAIST and also for his continuous assistance and support.

I am also grateful to all who have affected or suggested this area of research. Prof. Makoto Yokoo, at the Faculty of Information Science and Electrical Engineering, Kyushu University, and Associate Prof. Katsutoshi Hirayama, at the Faculty of Maritime Sciences, Kobe University, for their valuable suggestions and kind encouragements.

I would like to devote my sincere thanks and appreciation to my Japanese friends Dr. Saori KAWASAKI and Dr. Tokuro Matsuo for their sympathy and their unlimited help. My sin-

cere thanks goes also to all members of the Laboratory of Knowledge Creating Methodology.

A special word of gratitude to Dr. Shafeeque Ansari and his wife Dr. Zoubaida Ansari, to Dr. Mohamed Mostafa and his wife Maha, to Dr. Yasser Kotb, and to Dr. Rami Yared, for their kind help, encouragement and friendship.

I gratefully recognize all my Professors, in the High Institute of Management of Tunisia (ISG), Prof. Khaled Ghédira, of the High Institute of Computer Science of Tunis, and all my friends of the UR. SOIE Laboratory ISG Tunisia.

Last but by no means least; I am deeply grateful to all the members of the jury for accepting to judge this thesis.

To Whom this work is dedicated?

To my father Mohamed, for all his lavished love, trust, encouragement, and support, I am forever deeply grateful to him for all what he has done and still doing for me.

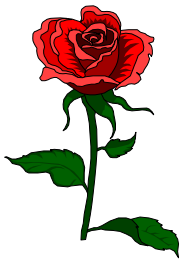
To my wonderful, adorable and devoted mother Essia who had never ceased taking care on me, even while I am abroad,

To my Dearest brother Naoufel,

To my brother Mourad and his wife Wafa and their two handsome kids Rami and Fares,

To my adorable sister Ibtissem and her husband Mohamed Ali and to the most pretty and wonderful girls, my two princess Ranime and Marame.

To my brother Wassim who never forgot to send me a nice card and a marvellous gift in my birthday.



To my dearest Samia for her support, kind help and deep love, with whom I shared, discovered and enjoyed the study and life in Japan,
to Dr. Nebil Achour and his wife my dear Caroline Deegan for being an excellent brother and sister for me and for their continuous help and encouragements.

To all the members of my family and to all my friends in Tunisia and Japan and all over the world (Canada, NewWork, France, etc.),

Finally to all those who encouraged me to persevere in the most difficult moments, that they find here the deep expression of my high gratitude.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context and motivation

Many combinatorial applications in real-world, known as NP-Complete problems, need to be solved. Several formalisms dealing with such problems were proposed in the literature, among which: linear programming problem formalism (PLNE), propositional satisfiability problem formalism (SAT), constraint satisfaction problem formalism (CSP).

The constraint satisfaction problem (CSP) formalism [67] is widely used to formulate and solve several combinatorial problems, e.g., planning, resource allocation, time tabling and scheduling. The great success of this paradigm is due essentially to its natural expressiveness of real-world applications. A CSP is defined by a set of variables, a domain of values for each variable and a set of constraints between these variables. Solving a CSP involves finding assignments of values to variables that satisfy all the constraints.

In instance, let's consider a simple real application, the map-coloring problem shown in Figure1.1. Assume that we have a map formed by four regions and we have only three colors to use for this map (*red*, *blue* and *green*). Our goal is to color each region in the map so that no adjacent regions have the same color and also the sea-side should not be colored in blue (assume that only one of the four regions is located near the sea). This problem can be easily formulated as a CSP in which, we have four variables $\{X_1, X_2, X_3, X_4\}$, each variable depicts one region. The domain of each variable is defined by the available colors $\{$*red*, *blue*, *green*$\}$. Two constraints occurs in this problem; The first one does not allow the use of the same color for each pair of variables corresponding to two adjacent regions. The second constraint is that the color used for the variable X_1, which is assumed to be the region near the sea, should not be blue. Solving this problem is assigning colors to variables (regions) with regards to the two constraints mentioned before.

Solving a CSP is a hard task and a blind search often leads to a combinatorial explosion in the search tree. However, this framework is marked by the ubiquitous use of local consistency (LC) properties and enforcing techniques; noting that LC is a relaxation of consistency. For any consistent CSP *P* there is a unique equivalent locally consistent and more simple

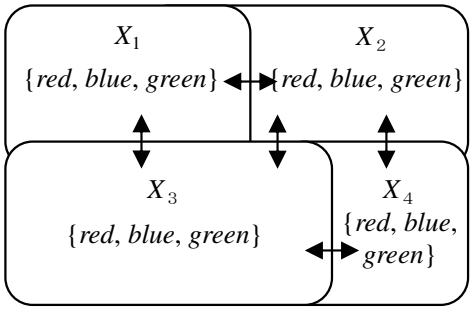


Figure 1.1. A simple example of the map-coloring problem. Three possible colors can be used for each region $\{red, blue, green\}$. Each arrow depicts two adjacent regions that should be painted with different colors. The region X_1 is supposed to be in the sea-side.

CSP *P'*. Finding *P'* is achievable in polynomial time by so-called enforcing or filtering algorithms. These algorithms allow the simplification of any constraint problems by eliminating values or combination of values that cannot be involved in any solution. In instance, in the above map-coloring problem, the variable X_1 should be different from *blue* according to the second constraint. For that reason, the value *blue* in the domain of X_1 cannot belong to any solution of the problem. Therefore, the value $blue$ can be removed from the domain of X_1 without loss of solutions.

Integrating the enforcing of local consistency as a preprocessing step and/or within the search process is worthwhile for pruning inconsistent values, consequently saving much fruitless exploration of the search tree especially on hard and large problems. Several levels of local consistency (node, arc, path and *k*-consistency) have been proposed in the literature. Obviously, as indicated in [31], the overhead caused by removing inconsistency has to be outweighed by its gain. This overhead fluctuates according to the problem to solve.

***Which level should be enforced when seeking for solutions in a constraint network*?**

Two main criteria should be taken into consideration while choosing a suitable level of consistency to achieve for a constraint network (CN). The first is the pruning efficiency of the filtering involved and, the second is its time and space complexities.

Arc consistency (AC) is the widely preeminent existing level of local consistency because it eliminates some values that cannot belong to any solution with a low cost. Enforcing AC embodies checking the consistency among each pair of variables connected by a constraint. This framework has been widely studied in many research efforts. The main reason is that maintaining AC during a search has been definitively shown to be a worthwhile approach when solving hard and large problems [7, 48].

There are two kinds of approaches to achieve AC, centralized and distributed approaches, both of them can be applied to *binary* CN and non-binary CN. In the binary CN, each con-

straint involves at most two variables while in the *non-binary* CN (called also *n-ary* CN or *general* CN) there is at least one constraint that implies more than two variables.

Some typical works in the centralized framework where discussed in the literature, such as in [104, 65, 71, 30], and [6]. As mentioned by Baudot and Deville [3], very few works deling with distributed approach can be found in the literature [87, 21, 78, 53], despite the natural distribution of many real-world applications and the advent of both distributed computing and networking technologies.

It is noteworthy that most efforts in constraint satisfaction problems assume that any real-life application can be exclusively formulated using binary constraints. Many of the academic problems amongst: n-queen, zebra, fit this condition, whilst for other real-application, their formulation requires imperatively the use of non-binary constraints in order to preserve problem semantic. Nevertheless, most efforts were devoted only to *binary* CN. The main reason is that any non-binary problem can be transformed into a binary one. Thus, many methods have been proposed in literature to translate non-binary constraints into an equivalent set of binary ones. Theoretically, this equivalence solves the issues of algorithms for non-binary problems. However, in practice, this translation presents several limitations concerning spacial and temporal requirements, which make it inapplicable. Furthermore, in [84] the author proved that this transformation could lead to the loss of a part of the constraints' semantics.

Recently further efforts have been devoted to extend binary techniques to non-binary versions able to deal with general constraints in their original form. However, only few works on enforcing arc-consistency for non-binary problems can be found in the literature [66, 72, 84, 11]. All these works address a centralized framework; no distributed approaches were suggested in the literature.

***Is AC enough for hard CN*?**

Performing only AC for some hard CNs might be fruitless; Case of problems initially AC. Consequently, applying this property may not delete any values, or may delete only few inconsistent ones. Therefore in achieving more local consistency pruning levels, k-consistency ($k > 2$), can be more efficient.

Higher consistency levels such as, path consistency, k-consistency, can prune more non-viable values. Some works dealing with enforcing path consistency (PC) were proposed in [31, 23]. These techniques check the consistency among all possible paths of three variables connected by three constraints in a complete CN. However, these techniques are never used in practice because of their very high complexities (or they are used only for very small and easy problems). Furthermore, enforcing k-consistency ($k \geq 3$) may change the graph of constraints[1] and especially require high computational cost.

[1] As mentioned in [103], these levels require the recording of forbidden tuples, which implies either the creation of new extensionally defined constraints, or the addition of an extensional definition to existing possibly

***What about a level higher than AC and less than PC*?**

Obviously, we should find a suitable level of consistency to achieve while considering the best compromise between the cost of the filtering process and the efficiency of the deletions involved. In [5] the author proposed the restricted path consistency (RPC) property, which is higher than the AC property and requires much less computational effort than PC. This level does not suffer from the drawbacks of PC. We notice also that no work has been proposed in the literature to enforce any level of local consistency, rather than AC, in an entirely distributed manner.

***What about tackling one of the hard real-world applications*?**

The great success of the filtering techniques in the enhancement of the solving process of many combinatorial problems, motivated us to tackle one among the hard real-world applications, which is the meeting scheduling problem. This problem is of great importance in our life and especially in the success of any organization. A good scheduling may lead a high gain for the organization and consequently to the society itself.

This problem embodies a decision-making process affecting several users, in which it is necessary to decide *when* and *where* one or more meeting(s) should be scheduled. To satisfy real-world efficiency requirements, in this work we focused on two challenging characteristics: the distributed and dynamic nature of the problem. The MS problem is inherently distributed and hence cannot be solved by a centralized approach; it is dynamic because users are frequently adding new meetings or removing scheduled ones from their calendar. This process often leads to a series of changes that must be continuously monitored.

The general task of solving an MS problem is normally time-consuming, iterative, and sometimes tedious, particularly when dealing with a dynamic environment. More precisely, solving the MS problem involves finding a compromise between all the attendants' meeting requirements[2] (i.e., date, time and duration) which are usually conflicting. Hence, this problem is subject to several restrictions, essentially related to the availability, calendars and preferences of each user. Automating meeting scheduling is important, not only because it can save human time and effort, but also because it can lead to more efficient and satisfying schedules within organizations [40].

Many significant research efforts were proposed in the literature among which [1, 4, 49, 96, 92, 63, 42]. Nevertheless, most of these works i) deal only with non-dynamic problems, ii) allow the relaxation of any user's preferences, iii) do not integrate the enforcement of local consistency in their solving process, iv) judge all the meetings of the whole system with the same level of importance, v) do not consider the high complexity of message passing operations in real distributed systems.

intentionally defined constraints.

[2]To simplify the problem, we assume that all the attendants are in the same city.

1.2 Objectives

We have learned from all the previous works and focused our research on the bellow points. Figure1.2 illustrates a summary the main objectives of this thesis.

- Propose new distributed hybrid method to enforce local consistency on any general CN. This new method involves two agent-based approaches. Those two approaches, called DRAC and GDRAC, are value-oriented propagation and concern distributed enforcement of AC for any binary and any general CN, respectively.
- Suggest a new solution to tackle higher level of consistency with reasonable complexities. The main idea is to propose a refinement of the DRAC approach to enforce more than AC with low cost, restricted path consistency (RPC), on any hard binary CN.
- Take the DRAC approach to the real world. We main of our third objective is to tackle one among the important combinatorial real-world application, the MS problem, while integrating filtering in the solving process. We focus essentially, in this work, on MS problem with predictable structure, i.e., All meetings are known in advance.
- Extend the protocol for solving any static MS problems to deal with unpredictable structure, i.e., case where the complete knowledge about whole problem is not available beforehand. Therefore, the underlying protocol must cope with all difficulties that may encounter with the dynamic environment requirements.
- Propose a new generic constraint-based asynchronous solver to deal directly with any constraint network.

New hybrid distributed method to enforce AC on any CN

For this point, the new hybrid method we suggest has the following characteristics:

- None of the approaches involved relies on any existing centralized algorithm.
- The underlying model, which is common for all the involved approaches, is based on a multi-agent system associating a reactive agent per constraint, each having a local goal. The full global goal of each approach is obtained as a result of the interactions between the reactive agents by exchanging asynchronous point-to-point messages containing inconsistent values.
- A dual constraint-graph is used to represent any CSP. This proposed model is different from the DisCSP [106] model, which is based on the primal representation of a CSP. The main objective is to be able to directly address any general CN without having any claim to any existing transformation non-binary $\leftrightarrow$ binary techniques. It is known that this transformation procedure may increase both the temporal and spatial complexity.

- The method can handle any kind of constraints, especially for the most important form defined by predicates for which no particular semantics is known.
- The global goal of each proposed approach in the system is accomplished with the minimum number of constraint checks and with the lowest CPU time required.

New agent-based approach to enforce more than AC on binary CN

For this second point, the new approach, called DRAC^{++}, does not rely also on any centralized techniques and it addresses especially hard CN where achieving only AC is ineffective.

New approach to solve any static MS problems

A new static multi-agent MS approach is proposed in this thesis. This approach closely reflects real applications while improving the process of scheduling meetings. The proposed protocol is based on distributed reinforcement for arc consistency (DRAC) approach. The basic idea is to benefit from the main goal of DRAC in order to reduce the complexity of a meeting-scheduling problem solving process. In this work, we propose to formalize the MS problem as a valued constraint satisfaction problem (VCSP) [36] in which each user maintains two kinds of constraints: *hard* and *soft* constraints related to them besides the other strong constraints defining the problem. The hard constraints (which can never be violated) represent the non-availability of the user, while the soft constraints (which can be violated) represent the preference calendar of a user. Furthermore, each new scheduled event is considered as a hard constraint.

More sophisticated and flexible solution to solve any dynamic MS problems

Another more sophisticated MS solver is proposed in this thesis. We have also adopted the agent-based model to this approach, because it is the most congruent system for a rich class of decision-making real-world problems. The MSRAC (Meeting Scheduling with Reinforcement of Arc Consistency), multi-agent coordination approach is a novel, scales better, dynamic and entirely distributed solution to the meeting scheduling problem that accounts for user preferences, handles several events with various levels of importance and especially minimizes the number of exchanged messages. The basic characteristics of MSRAC are the following.

- First, it is an incremental approach capable of processing problem alterations without conducting any exhaustive search.
- Second, it is based on the DRAC approach to enhance the efficiency of the solving process.

- Third, in the MSRAC approach the MS problem is contemplated as a set of distributed reactive self-interested agents in communication, each with the ability to make local decisions on behalf of the user. The agents' decisions are not based on any global view[3] but only on currently available local knowledge. The final result is obtained as a consequence of their interactions. This purpose is achieved with the minimum number of exchanged messages by virtue of the real difficulty of message passing operations in a distributed systems.

- Finally, the use of preferences naturally implies the adoption of an optimization criterion, both for each agent and also for the system as a whole. Thus, we adopted the dynamic valued constraint satisfaction problem formalism (DVCSP) to model any MS problems. This formalism provides a useful framework for investigating how agents can coordinate their decision-making in such dynamic environment leading to more flexible and widely applicable approach to real-life.

New generic asynchronous approach to solve any distributed constraint problem

As for our main contribution in the fith point, is to propose a novel complete and generic multi-agent algorithm for any CN. The new approach is able to solve any CSP while performing distributed enforcement of AC, without adding any new links and without recording any nogoods. The main reason for using a lazy version of DRAC is to save some fruitless backtracking and consequently to enhance the efficiency of the proposed approach.

In addition, we propose a generic distributed method to compute a static constraint ordering in which we save as many links as possible in order to decrease the set of exchanged messages. Furthermore, information about variables may belong to different agents while information about constraints belongs only to the owner agent and is kept confidential.

1.3 Thesis guideline

This thesis is divided into ten chapters.

Chapter 2 introduces some useful definitions and proposed techniques for the constraint satisfaction problem formalism and its extensions.

Chapter 3 presents some useful definitions of local consistency property and discusses some of the existing centralized and distributed enforcement techniques.

[3]The agents exchange as little information as possible to keep most of their personal information private.

Chapter 4 defines one of the real world application, meeting scheduling, with a review of some of the related works.

Chapter 5 introduces a novel hybrid agent-based method including the two following approaches, to enforce AC on binary CN, DRAC approach, and for n-ary CN, G-DRAC approach.

Chapter 6 presents an agent-based approach to enforce more than arc consistency on binary and hard constraint network, DRAC^{++} for distributed restricted path consistency enforcement.

Chapter 7 illustrates a novel approach to solve any static meeting scheduling problem, MSS (for *MS Solver*).

Chapter 8 introduces a more sophisticated solver for any dynamic MS problem, to cope with encountered difficulties in the dynamic environment, MSRAC (*Meeting Scheduling with Reinforcement of Arc-Consistency*).

Chapter 9 discusses a generic, new distributed, complete and constraint-based approach to solve any distributed problems, DisAS (for *Distributed Asynchronous Search*).

Finally, **Chapter 10** concludes the thesis.

1.4 Notations and conventions

The abbreviations used in this thesis are summarized in the following table:

CSP	Constraint satisfaction problem
SAT	Satisfiability problem
CN	Constraint network
DisCSP	Distributed constraint satisfaction problem
VCSP	Valued constraint satisfaction problem
DCSP	Dynamic Constraint Satisfaction problem
DynVCSP	Dynamic valued constraint satisfaction problem formalism
BT	Backtracking
AWC Search	Asynchronous weak-commitment search
AAS	Asynchronous aggregation search
LC	Local consistency
NC	Node consistency
AC	Arc conssitency
PC	Path consistency
RPC	Restricted path consistency
CT	Clarck Tax mechanism
MS	Meeting scheduling
MAS	Multi-agent system
DRAC	Distributed reinforcement of arc consistency
G-DRAC	General distributed reinforcement of arc consistency
MSRAC	Meeting scheduling with reinforcement of arc consistency
MSS	Meeting scheduling solver
DisAS	Distributed asynchronous solver
GU	Global utility
LU	Local utility

The most used notations are as follows:

A_i	Agent number i
X_i	Variable number i
$\mathrm{D}(\mathrm{X}_i)$	Domain of the variable number i
$\mathrm{C}_{ij...}$	Non-binary constraint
$\mathrm{C}_{ij...}^{A_i}$	Non-binary constraint maintained by the agent i
C_{ij}	Binary constraint involving only two variables X_i and X_j
$\mathrm{R}_{ij...}$	Relation associated to the constraint $\mathrm{C}_{ij...}$
Const(X_i)	Set of constraints involving of variable X_i
Var($\mathrm{C}_{ij...}$)	Set of variables involved in the constraint $\mathrm{C}_{ij...}$
t	Vector of viable variables/values
v_{i_l}	the l^th Value in the domain of the variable X_i
index($\mathrm{C}_{ij...}$, X_k)	Position of the variable X_k in the constraint $\mathrm{C}_{ij...}$
Γ	Set of all Constraint agents in the system
Γ^{A_i}	Set of acquaintance of the agent A_i
TupleSupport^{A_i}	Set of tuples allowed by the constraint associated to A_i
m_h^i	h^{th} meeting of the agent A_i
dt_p	p^{th} date in the domain
C_s	Soft constraint
C_h	Hard constraint
$w_p^{A_i}$	Degree of preference of the agent A_i to schedule meeting at the p^{th} date
W_{X_k}	Importance of the k^{th} meeting

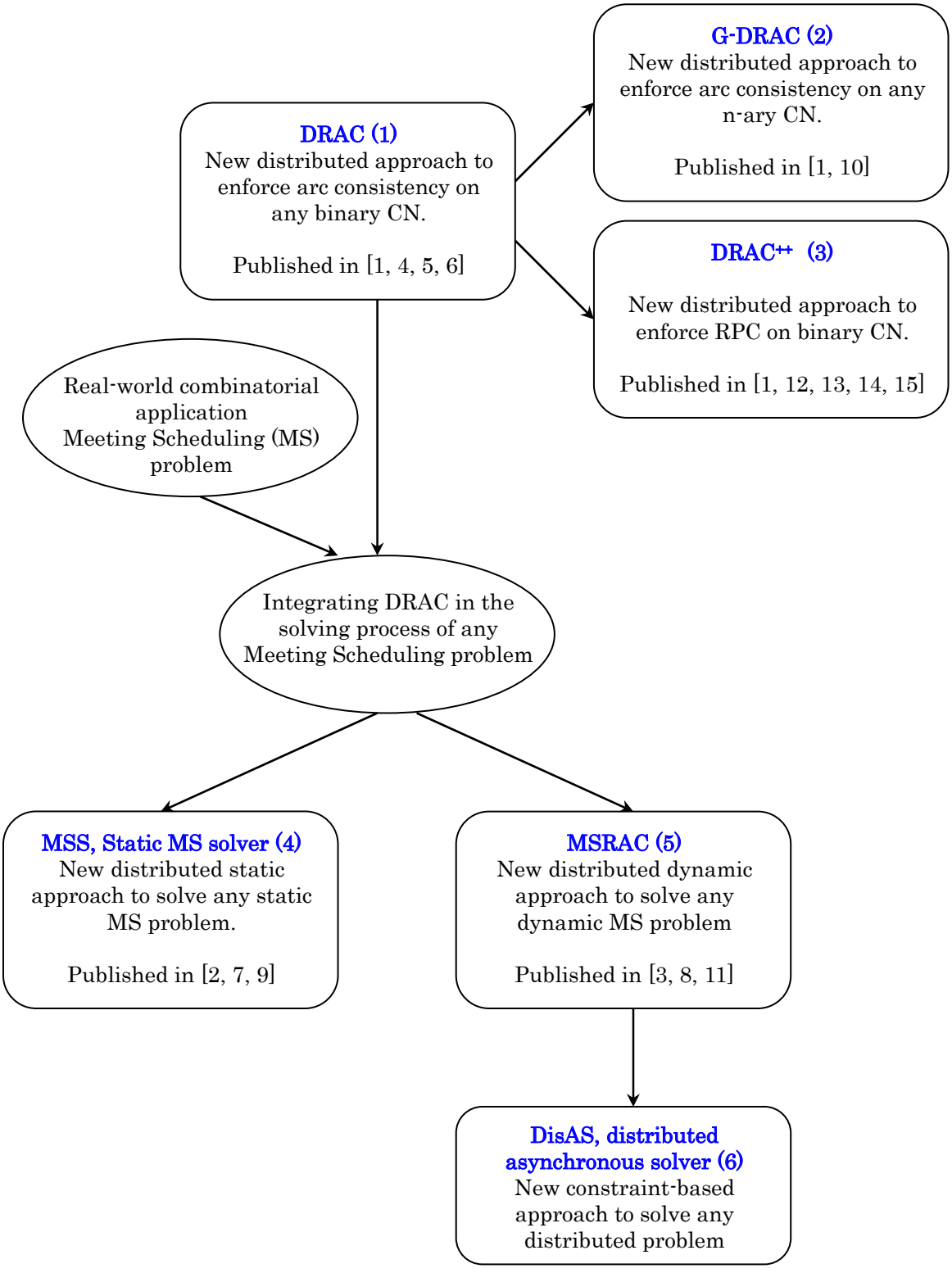


Figure 1.2. A summary of the main objectives of this thesis and the underlying publications.

Chapter 2

Constraint Satisfaction Problem Formalism

Constraint satisfaction problem (CSP) formalism [67] is widely used to formulate and to solve many combinatorial problems, such as planning, resource allocation, time tabling and scheduling. The great success of this formalism is due essentially to its natural expressiveness of real-world applications. Several ways of modeling a given problem as a CSP, are possible. Nevertheless, the choice of the model can have large impact on the required time to find solutions [76]. However, as mentioned by Bacchus et al. [16], besides the various possible modeling techniques that have been developed such that adding redundant and symmetry-breaking constraints [50, 94], adding hidden variables [28]. One important modeling decision is the arity of each used constraints, i.e., the number of variables involved in each constraint. A constraint can be expressed over a pair of variables, case of a binary constraint, or over a set of variables (more than two), case of a non-binary constraints.

In the sequel of this chapter, we will give first some basic definitions and notations for the CSP formalism and some of its extensions. Then we will describe some of the existing solvers.

2.1 Definitions and preliminaries

Definition 1 *Informally, a constraint satisfaction problem [67] (CSP) is a tuple (X, D, C) where:*

- *$X=\{X_1, \ldots, X_n\}$, is a finite set of **n** variables,*
- *$D=\{D(X_1), \ldots, D(X_n)\}$, is a set of **n** finite domains. $D(X_i)=\{v_{i_1}, \ldots, v_{i_d}\}$ with $|D(X_i)|=d$. A total order $<_d$ can be defined on the values of each domain, without loss of generality. For each pair of values $\{v_{i_k}, v_{i_l}\} \subseteq D(X_i)$, $v_{i_k} \prec_{lo} v_{i_l}$ if and only if $v_{i_k} < v_{i_l}$.*

- *$C=\{C_{ij...}, \ldots\}$ is a set of **e** constraints between these variables. Each constraint $C_{ij...}$ implies an ordered set of variables Var($C_{ij...}$)=$\{X_i, X_j, \ldots\}$. $|Var(C_{ij...})|$=**r** is the arity of the constraint. Let's denote by Const(X_i) the set of all constraints $C_{ij...}$ involving X_i while index($C_{ij...}$, X_k) is the position of variable X_k in $C_{ij...}$. $|Const(X_i)|$=m, **m** is called the degree of X_i. The constraints restrict the values of the r variables that can be simultaneously taken.*

  Each constraint $C_{ij...}$ can be represented implicitly by an arithmetic relation or by a predicate, where a computation is needed to check if the underlying constraint is satisfied or not. Or explicitly by the set of allowed (or forbidden) tuples (denoted by $R_{ij...}$), where the answer to a constraint check is already recorded in a database. The majority of works on constraint reasoning has focused on ways to reduce the number of constraint checks required in order to decrease the temporal complexity of the solver.

  An instantiation of the variables in Var($C_{ij...}$) is called a tuple on Var($C_{ij...}$). Assume that there are two tuples t_1 and t_2 on Var($C_{ij...}$). A lexicographical order $\prec_{lo}$ can be also set between the tuples on variables of a constraint $C_{ij...}$ in which $t_1 \prec_{lo} t_2$ if and only if it exists k such that t_1[1..k-1]=t_2[1..k-1] and $t_1[k] <_d t_2[k]$ (where t_1[1.. k-1] is the prefix of size k of t_1 and t_1[k] is the k^th value of t_1).

Definition 2 *A full or partial assignment $I_Y=\{v_{1_j}, v_{2_l}, \ldots, v_{m_p}\}$ is a vector of values such that every $v_{i_l} \in D(X_i)$.*

Example 1 Let's consider one of the most important problem of the general system for mobile communication GSM, which is the frequency assignment problem (called also channel assignment problem) [97]. Given a set of geographically divided, typically hexagonal regions called cells (see Figure2.1). Frequencies (channels) must be assigned to each cell according to the number of call requests. This problem has three types of electro-magnetic separation constraints:

1. Co-channel constraint: the same frequency cannot be assigned to a pairs of cells that are geographically close to each other.
2. Adjacent channel constraint: similar frequencies cannot be simultaneously assigned to adjacent cells.
3. Co-site constraint: any pair of frequencies assigned to the same cell must have a certain separation.

To solve this problem is to find a frequency assignment that satisfies the above mentioned constraints and while minimizing the sum over all co-channel and adjacent channel interferences.

A possible formulation of this problem is as given by Sivarajan et al. [97] where frequencies are represented by positive integers 1,2,3, . . .
Given*:*

- N, the number of cells,
- d_i, $i \in \{1, .., N\}$, the number of requested calls (demands) in cell i,
- C_{ij}, $1 \leq i, j \leq N$, the frequency separation required between a cell in cell i and a call in cell j.

We need to find: f_{ik} the frequency assigned to the k^{th} call in cell i with $1 \leq i \leq N$ and $1 \leq k \leq d_i$, such that $|f_{ik} - f_{jl}| \geq C_{ij}$ for all i, j, k, l except i=j and k=l and while $\min \max f_{ik}$ for all i, k.

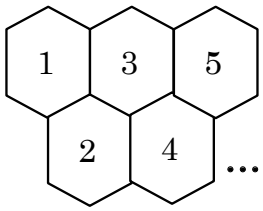


Figure 2.1. Example of the hexagonal regions used in the frequency assignment problem.

Definition 3 *Let (X, D, C) be a CSP, A solution of the CSP is defined by the ordered set of variables X and an assignment I_X of a value to each variable in X satisfying all the constraints in C.*

I_X=*{(X_i, v_{i_k})| ∀ $X_i \in X$ and ∀ $C_{ij...} \in C$ / $X_i \in Var(C_{ij...})$; (X_i, v_{i_k}) satisfies all Const(X_i).}*

Solving a CSP consists in finding one or all full assignments. This type of problems is known as NP-Complete for which the solving task is hard, where a blind search often leads to a combinatorial explosion. The NP-complete problems are the most difficult problems in NP.

Definition 4 *NP is the class of problems for which a claimed solution can be tested within a polynomial time on the length of the problem description.*

Definition 5 *A NP-complete problems [43] is a subclass of NP problems to which a SAT problem can be mapped within a polynomial time bounded by the length of the problem description.*

Definition 6 *A binary CSP is a problem where all the constraints are binary constraints; otherwise is it called n-ary CSP.*

Definition 7 *A constraint is a binary constraint if and only if it involves at most two variables; otherwise the constraint is called non-binary (n-ary constraint).*

Following Montanari [67], a binary relation corresponding to a constraint C_{ij} between two variables X_i and X_l can be represented by a (0, 1)-matrix with $|D(X_i)|$ rows and $|D(X_l)|$ columns by imposing an order on the domains of the variables. A *zero* entry at row a column *b* means that the pair consisting of the a^{th} element of $D(X_i)$ and the b^{th} element of $D(X_l)$ is not permitted; a *one* entry means that the pair is permitted. However for the case of constraint in intension, to determine all the allowed couples of values requires high time and space cost.

Definition 8 *A binary relation R_{ij} corresponding to a constraint C_{ij} represented as a (0, 1)-matrix is row convex if and only if in each row all of the ones are consecutive; that is, no two ones within a single row are separated by a zero in that same row.*

Consider the example given in Figure 2.2, the binary relation C_{12} between X_1 and X_2 is row convex relation.

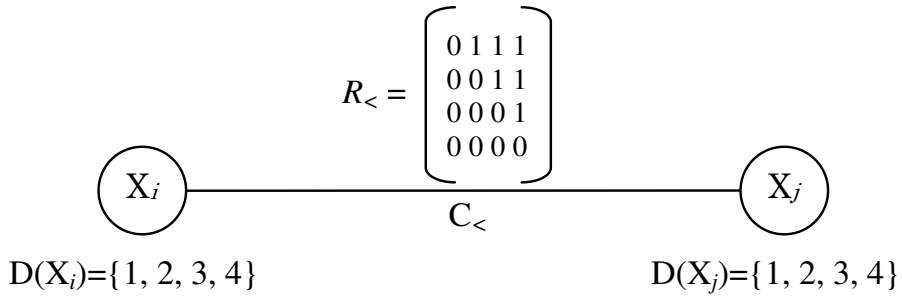


Figure 2.2. Example of a row-convex binary relation.

We give now the definition of a special constraint, *all-different* constraint, which will be used throughout this thesis.

Definition 9 *A constraint $C_{ij\ldots}$ on variables $\{X_{i_1}, X_{i_2}, \ldots, X_{i_r}\}$ with r is the arity of the constraint, is called an* ***all-different*** *constraint [90] if and only if it allows the tuple (a_1, a_2, $\ldots$, a_r) $\in D(X_{i_1}) \times D(X_{i_2}) \times \ldots \times D(X_{i_r})$ such that $a_k \in D(X_{i_k})$ and for all $l \neq m$, $a_l \neq a_m$.*

Three graphic representations can be used to represent a CSP: primal graph, dual graph [27] and hypergraph [100].

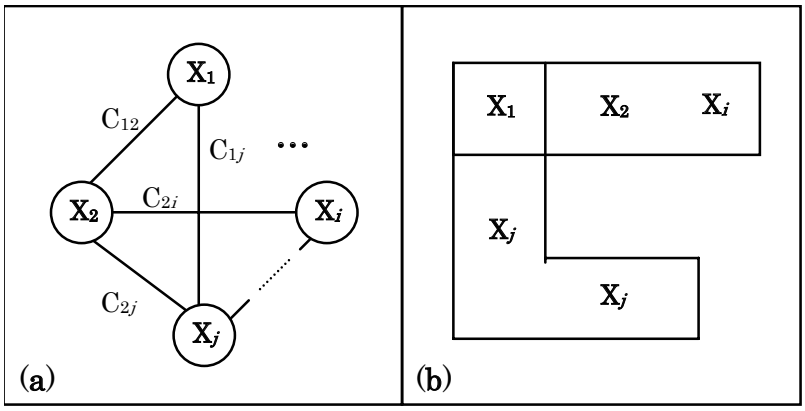


Figure 2.3. Example of a primal graph for a binary constraint problem (a) and a non-binary constraint problem (b). The nodes represent the variables while the (hyper)-links illustrate the common constraints.

- The primal graph: a CSP is represented as a graph where the nodes are the variables of the underlying problem and the links are the constraints. Each pair of variables X_i and X_j are linked together if and only if they share at least one constraints (see Figure 2.3(a)).

- The dual graph: this representation comes from the relational database community and was introduced to the CSP community by Dechter an Pearl [27]. In this representation the constraints labeled the nodes of the graph, and the variables labeled the links relating the nodes (see Figure 2.4).

- The hypergraph: This graph is used to represent non-binary constraints. The variables labeled the nodes of the graph and the hyper-links represent the n-ary constraints (see Figure 2.3(b)).

However with the advent of both distributed computing and networking technologies, many naturally distributed problems arise leading to the birth of a new subfield of the AI, the distributed AI (DAI). This new subfield requires a new formalism to develop a general framework for DAI. The distributed constraint satisfaction problem (DisCSP) is an extension of the CSP formalism [106] to represent a variety of distributed problems where constraints and/or variables are controlled by a set of independent but communicating agents, such as distributed resource allocation problem [22], distributed scheduling problem [95], multi-agent truth maintenance tasks [55].

Definition 10 *A distributed constraint satisfaction problem (DisCSP) [106] is a CSP whose variables and constraints are distributed among multiple agents.*

- *There exist n agents 1, 2, . . . , n.*

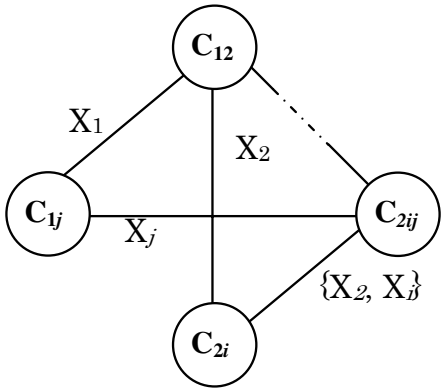


Figure 2.4. Example of a dual graph for any constraint problem. The nodes represent the n-ary constraints while the links define the shared variables.

- *Each agent has several variables.*
- *Each agent i knows all constraint predicates relevant to its variables (constraint predicates which take i's variables as arguments).*

Another extension of the CSP formalism was also proposed in the literature, mainly to represent some real-life scenarios where it is impossible to satisfy all the constraints. In, this case known as over-constrained problems, we may allow the relaxation of some constraints to solve it. The proposed valued constraint satisfaction problem (VCSP) formalism consists of giving a weight or a valuation to each constraint to reflect the importance of satisfying it.

Definition 11 *A valued constraint satisfaction problem (VCSP) [88] is a quintuple (X, D, C, S, φ) where (X, D, C) is the classical CSP formalism, S= (E, $\otimes, \succ$) is a valuation structure, and $\varphi : C \rightarrow E$. E is the set of possible valuations; $\succ$ is a total order on E; $\bot \in E$ corresponds to the maximal satisfaction and $\top \in E$ corresponds to the maximal dissatisfaction; $\otimes$ is an aggregation operator used to aggregate valuation. Assume that A is an assignment of all the variables of the problem. The valuation of A is defined by φ(A)= $\otimes_{c \in C} \varphi$(A, c) where*

$$\varphi(A, c) = \begin{cases} \bot, & \textit{if c is satisfied by A;} \\ \varphi, & \textit{otherwise.} \end{cases} \tag{2.1}$$

In the aforementioned formalisms, CSP and VCSP, the knowledge about the problem is assumed to be totally known and fixed. However, this is not always possible especially when dealing with real situation where the underlying problem may evolve in time due to *i*) the environment, evolution of the set of tasks to be perform and/or of their execution conditions in scheduling applications; *ii*) the user, evolution of the user requirements in the framework of an interactive design; and *iii*) the other agents is the framework of a distributed system.

The notion of dynamic CSP (DCSP) [26] has been introduced to represent such situations.

Definition 12 *A* dynamic *constraint satisfaction problem* P *(DCSP) [26] is a sequence of static CSP* P_0, ..., P_a, P_{a+1}, ... *each resulting from a restriction (a constraint or a variable is added) or relaxation (a constraint or a variable is retracted) in the preceding one.*

Several techniques to solve constraint satisfaction problems were proposed in the literature. These techniques can be divided into several categories: centralized and distributed, complete and incomplete, synchronous and asynchronous, etc. In the following we will present some of them.

Definition 13 *An algorithm is complete if and only if it guarantees to find a solution, if one exists, or to prove that the problem is insoluble, otherwise.*

Definition 14 *Let (X, D, C) be a CSP, A partial solution to the CSP is defined by a ordered subset of variables* $Y \subseteq X$ *and an assignment* I_Y *of a value to each variable in Y.*

2.2 Constraint reasoning techniques

Two types of real-world applications can raise according to their physical location. The first concerns the traditional centralized problems, where all the data is gathered on the same site. The second kind deals with the naturally distributed problems among several sites and for which it is not convenient to gather the whole problem knowledge into a single site. The main reason and not the only one is the cost of collecting all information into the same site could be taxing. Furthermore, gathering all information into a single site could be undesirable essentially for security or privacy reasons. Our research were motivated by the second type besides its importance and frequency in our real life. However for centralized problems, the large variety of existing centralized techniques are worthwhile to solve them. Whilst, for the second type of problems we need to apply a distributed techniques. Hence, during last few years AI community has shown an increasing interest in solving such problems using multi-agent system (MAS) paradigm. Moreover, note that even for some centralized problems applying distributed techniques is better and this for security reason [51]. These problems are known as artificially distributed problems.

In the sequel of this chapter, we will review the two existing types of constraint programming techniques, centralized techniques and parallel/distributed techniques.

2.2.1 Centralized search

Two main groups of centralized CSP solvers exist in the literature: The search algorithms and the consistency algorithms. The former can be divided into two groups Backtracking algorithms and iterative improvement algorithms. However for the consistency algorithm, they

can be used as preprocessing techniques or during the search process to reduce futile backtracking and consequently to enhance the efficiency of the search process. These techniques will be given in detail in Chapter 3.

Backtracking algorithms

Chronological Backtracking (BT) algorithm [45] is the basic for most systematic algorithms for solving CSPs. Such algorithm is known to be complete, it proceeds first by constructing a partial solution including a value assignment of a subset of variables $Y \subseteq X$ that satisfies all the constraints within Y. This partial solution will be extended progressively to a complete solution (if possible) by adding new variables (the next in the ordering) one by one until exploring all the variables of the underlying problem. A dead-end is detected when for one variable X_i, no possible value, satisfying all the constraints between X_i and the partial solution, is found. In this case, the value of the most recently added variable X_j to the partial solution is changed. This operation is called *backtracking* (BT). This algorithm terminates when all the variables have been assigned a value, in this case it returns this solution, or when all the variable-values combinations have been checked and failed, case of insoluble problem.

This algorithm is depth-first tree search algorithm where its efficiency is subject to enhancement. Several heuristics have been proposed in the literature to ameliorate the search process of the BT algorithm, such as the order of selecting variables, the order of selecting values, etc. The value-order heuristic known as min-conflict heuristic [73] is the most successful one among the existing ones. The basic of min-conflict BT consists in choosing the value that satisfies as many constraints with the tentative variables in the partial solution.

Several complete centralized enhanced search algorithms based on backtracking have been proposed in the literature for binary CSPs.

Backjumping (BJ) [46] is more intelligent than BT in the way to behave when a dead-end occurs. This algorithm avoids the computational overhead of BT by using syntactic methods to estimate the point to which BT is necessary. Instead of backtracking to the previous variable, it backjumps to the deepest past variable X_k ($\mathrm{X}_k \prec \mathrm{X}_i$) in conflict with the current variable X_i. In this way BJ avoids redundant reassignment of values to any variables X_l with $\mathrm{X}_k \prec \mathrm{X}_l \prec \mathrm{X}_i$ since these variables are not involved in the detected conflict. However, BJ needs to record the deepest conflicting variable X_k for each X_i in order to avoid the re-exploration of the dead-end branch of the search tree.

Conflict-direct backjumping (CBJ) [80] is a refinement of BJ while doing more sophisticated backjumping. This algorithm proceeds by recording the set of conflicting past variables X_k for each variable X_i. Therefore, it requires more complicated data structure that will be used in case of dead-end to perform a backjump not to the source of conflict but to the con-

flicting variable closest to the root of the search tree, i.e., the deepest variable in its conflict set. Hence, in case of conflict CBJ would jump to further position in the search tree compared to BJ.

Backmarking [47] (BM) is another refinement of backtracking algorithm based on marking scheme process. This process saves many redundant consistency checks in order to avoid repeating them. When the instantiation of two variables have not changed since last time they were checked, they will be marked and this information will be recorded in special data structures to avoid checking them again. Other enhancements of BM algorithm were proposed in the literature, amongst: backmarking with conflict-directed backjumping [80] (BM-CBJ), BM-CBJ2 [59].

Forward checking [56] (FC) is a look-ahead algorithm. It checks the current assignment against all future variables/values that are connected to the current variable X_i. Each inconsistent value belonging to a future variable X_j is temporarily removed from the domain of X_j. If a domain of a future variable X_j becomes empty, the instantiation of the current variable is undone, and another value is tried. If no possible other value is found for the current variable then a backtrack is performed. FC guarantees for each current partial solution the consistency of the current value with the already assigned past variables (by construction they are consistent and no need to check them again). Several extensions of FC were proposed in the literature, amongst: FC-CBJ [80], FC-BM [81], Minimal FC [25].

Maintaining arc consistency (MAC) [86] is also a look-ahead algorithm. This algorithm performs more than lazy arc consistency. While checking the consistency of the current assignment with the connected future variables, this algorithm proceeds by enforcing arc consistency on the whole subproblem formed by all the future variables. This extra-work performed by MAC may delete more values from the domains of future variables and consequently may lead to more pruning in the search space compared to FC.

It is noteworthy that most of the backtracking algorithms deal only with binary CN. Some researchers claimed that their algorithms are also worthwhile for general CN, others provided an extension of their work to deal with non-binary cases amongst, the work done by Gent and Underwood [52]. In this work, the authors presented a general definition and implementation of CBJ for constraints of arbitrary arity. FC has been also generalized in a straightforward way to handle n-ary constraints [57]. Others and more stronger generalizations of FC to n-ary problems were introduced by Bessière et al. [12].

Iterative improvement algorithms

As described by Yokoo and Hirayama [107], these algorithms are based on hill-climbing search. An initial value is given randomly to each variable of the problem. The obtained configuration is then progressively revised by using hill-climbing search until finding a consistent solution, if it exists. The main limitation of these algorithms is to fall into local-minima rather than global one, case where some constraints are violated and the number of these violated constraints cannot be decreased by changing any single variable/value. Several techniques were proposed to escape from local-minima, for example in the breakout algorithm [74] a weight is defined for each constraint (initial weight is 1). The summation of the weights of violated constraints is used as an evaluation value. In case of local-minima, this algorithm increases the weights of violated constraints is the current configuration by 1. The evaluation value of this configuration will be larger than those of the neighboring configurations. Hence, the iterative improvement algorithms can be efficient, but their completeness cannot be guaranteed.

2.2.2 Parallel and distributed search

At this point we have to distinguish first between two main paradigms, parallel problem solving and distributed problem solving. According to Wooldridge [105], parallel problem solving merely involves the exploitation of parallelism in solving problems. The computational components are simply processors; a single node will be responsible for splitting up the overall problem into sub-components, allocating each of them to a processor, and subsequently assembling the solution. Nodes are assumed to be homogenous. In contrast a distributed system is defined by a set of entities sharing a common goal and thus there is no potential for conflict between them. The problem cannot be solved without cooperation. Cooperation is necessary between the entities because different nodes might have different parts of the problem.

The two main objectives behind distributing the solving process are, i) to speed up the running time of a central algorithm, or ii) to solve the problem that is already distributed among these entities and there is no way to gather all the information on the same node, i.e., for time/cost or for security reasons.

However, the main assumption made by most work on distributed problem solving concerning implicitly sharing the same goal for all the entities in the system, is worthwhile for entities belonging to the same organization or individual. In contrast, in multi-agent systems [105] paradigm (MAS), it is assumed instead that entities (called agents) are self-interested entities and they are concerned with their own welfare (of course on behalf of some users/owner).

In addition, some real applications require negotiations between the entities of the problem, such problems: meeting scheduling problems, distributed resource allocation problems,

etc. Hence, MAS concerns issues such as how agents can reach agreement through negotiation on matters of common interest, and how agents can dynamically coordinate and cooperate their local activities with other ones whose goals and motives are unknown. Several applications in real-world are concerned with finding a consistent combination of agent actions (e.g., distributed resource allocation problems [22], distributed scheduling problems [95], multi-agent truth maintenance tasks [55]). These problems can be naturally formalized as a DisCSP [111] and consequently can be solved using distributed algorithms. However, as mentioned by Yokoo [107] existing parallel/distributed algorithms for CSP are not worthwhile for DisCSP due to the fact that they usually require global knowledge/control among agents.

Another point should come up while talking about distributed system modeling. As indicated in [85], it is useful to distinguish between synchronous and asynchronous system. With asynchronous systems we have no assumptions about process execution speeds and/or message delivery delays; with synchronous systems we do make assumptions about these parameters. Hence, in asynchronous protocols, agents can proceed independently without explicit synchronization. Nevertheless, asynchronous gives agents more freedom in the way they can contribute to the search, and especially allowing them to enforce individual policies such as privacy. In this thesis we are interested by the proposed algorithms for DisCSP.

Mainly two types of models are used in most of the algorithms presented in the literature for solving constraint satisfaction problems in a distributed manner, the variable-based model and the constraint-based model.

Definition 15 *A variable-based model is a model where each variable belongs to one agent and constraints are shared between agents. A constraint-based model is a model where each constraint belongs to one agent and involved variables are shared between agents.*

In the following we briefly review some of the most important existing distributed techniques for solving distributed constraint problems.

Asynchronous backtracking (ABT)

This algorithm proposed by Yokoo et al. [106] is a distributed version of the backtracking algorithm to solve DisCSP. This algorithm assumes a variable-based model. Let's recall that for this model, constraints and variables of the problem are distributed among a set of automated agents. Each agent is responsible for maintaining one variable[1]. it has a link toward any agent that owns a constraint involving that variable. Agents are arranged in a fixed priority order $\succ$, i.e., $A_i \succ A_j$ if and only if $i < j$. A constraint is enforced by the lowest priority agent among those that are responsible for the variables in the constraints. ABT is executed autonomously and asynchronously by each agent in the network. It computes a global consistent solution (or detects that no solution exists) in finite time.

[1] Although ABT can be applied to the situation where one agent has multiple local variables.

Each agent instantiates randomly its variable and communicates the value to the relevant agents (connected by outgoing links) via *ok?* message. No agent has to wait other agents' decisions. Each agent that received an *ok*? message with varaiable/value assignment, evaluates its constraints involving received variables. If the evaluation process succeeds, i.e., all constraints involving this variable are satisfied with the new assignment, do nothing. Otherwise, the concerned agent will try to assign a new value for its variable if possible. If no other viable value is found, the agent generates a *nogood* message and send it to the lowest priority agent generating a dead-end assignment, i.e., an assignment that cannot be extended to a complete solution. The agent receiving this nogood message will incorporate this information in its local knowledge, change its current assignment if possible, otherwise generate another *nogood* message accordingly. The local knowledge of an agent A_i is formed by its own agent view and a set of nogoods. The agent view of A_i is a set of values that it believes to be assigned to agents connected to it by incoming links while the set of nogoods is kept as a justificative of inconsistent value. The process terminates when achieving quiescence, meaning that a solution has been found, or when the empty nogood is generated, meaning that the problem is unsolvable. The completeness of this algorithm is given by Yokoo et al. [109].

Asynchronous weak-commitment search (AWC Search)

In the previous algorithm, ABT algorithm, lower priority agents need to make an exhaustive search to revise bad decision(s) made by higher priority agent(s). Therefore, Yokoo [110] proposed an extension of ABT algorithm based on both, the min-conflict heuristic to reduce the risk of making bad decisions, and the dynamic agent ordering in order to be able to revise bad decisions without conducting and exhaustive search. Hence, for the asynchronous weak-commitment search algorithm (AWC), the priority value is determined for each variable and communicated to other agents via *ok?* message. The priority order is determined by the communicated priority values, i.e., the agent/variable with the larger priority value has the higher priority. In case of conflict, the current value of the agent is inconsistent with the received assignment, the agent choose another value using the min-conflict heuristic, i.e., choose the value that minimizes the number of violated constraints.

Each agent that cannot instantiate its variable, i.e., no consistent value is found, sends a *nogood* message to the nearest higher priority agent and increases its priority value. However if the agent cannot generate a new nogood, it will not change its priority value but will wait for the next message. This process is used in order to guarantee the completeness of the algorithm. The author provides in [110] a proof of the completeness of the AWC algorithm. However, as mentioned by Maestre and Bessière [70], this algorithm is incomplete unless agents can store a potentially exponential number of nogoods.

Distributed backtracking (DIBT)

The basic of this algorithm is the backtracking algorithm (BT) [45]. The authors proposed [54] a new ordering schema that fits the constraint graph topology to take advantage of the features of the problem. This order is dissimilar to the lexicographic ordering of agents used in ABT. They proposed a generic method for distributed computation of any static variable ordering according to a chosen heuristic, e.g., *max-degree* heuristic that chooses higher order of the variable involved in maximum number of constraint.

The authors perform an exhaustive domain exploration to ensure the completeness of DIBT. In their algorithm, the author avoided learning schemes, such as nogood recording. the constraint checks are parallelized and whole system operates in a conservative strategy for saving benefits of previous search in independent parts of the network. The authors used an ordering that fits the constraint graph topology, which allows a free graph-based back-jumping behavior during failure phases. However, DIBT is not complete.

Asynchronous aggregation search (AAS)

Another extension of ABT algorithm was proposed by Silaghi et al. [98] where constraints can be private knowledge of some agents and several agents are allowed to simultaneously propose instantiations for the same shared variable. The authors propose to integrate the aggregation process of tuples to enhance the efficiency of their algorithm, the asynchronous aggregation search (AAS). The authors propose three different variant of this technique based respectively on full, partial and no nogoods recording.

This work is considered as an ABT for dual graph. The basic idea of AAS technique consists in propagating aggregated tuples of Cartesian product of values rather than individual values themselves. The agents are assigned static priority basically based on the lexicographic order. A link is set between each pair of agents if they share at least one variable. AAS works in exactly the same manner as ABT, except that messages refer to Cartesian products. If an agent find no combination in the Cartesian product $\{X_i=\{a_1, \ldots, a_l\}\}\times\{X_j=\{b_1, \ldots, b_k\}\}$ is compatible with its constraints, it generates a nogood for this combination and sends it to a higher order agent. The result of the search is no longer a list of individual assignments but a set of domains whose Cartesian product contains only solutions. The authors claim that several techniques can be used for aggregation, and that these techniques are sound and terminate in a finite time. However, aggregation process might be expensive in terms of constraint checks.

Asynchronous backtracking without links ABT$_{not}$

This algorithm is a new member in the ABT family proposed in [13]. It is an ABT-based algorithm that does not require the addition of communication links between initially unconnected agents. This algorithm proceeds like ABT without requiring additing of new links.

The authors propose first the ABT$_{kernel}$ algorithm, which requires like ABT, that constraints should be directed from the constraint-sending agent to constraint-evaluating agent forming a directed acyclic graph. In this algorithm, every new nogood is obtained as a conjunction of stored nogoods sharing the faulty variable. However, this algorithm is sound but may fail to terminate due to obsolete nogoods.

Hence, in their work, the author proposed several alternatives to rely to this limitation. Adding new links can be performed under several conditions, adding new permanent links as preprocessing (ABT$_{all}$ algorithm), adding new permanent links during search (ABT algorithm), adding temporary links (ABT$_{temp}$), i.e., these links will be removed after a fixed number of messages, without adding any links (ABT$_{not}$), i.e., in case of failure the agent backtracks and forgets all the nogoods that hypothetically may become obsolete. Nevertheless, as longer as the obsolete nogoods remain in the local knowledge of an agent, it will absolutely affect the efficiency of the constraint solver.

2.3 Summary

In this chapter we presented an overview of research related to the constraint satisfaction problem formalism and its extensions. We gave some useful definitions to allow a better understanding of this paradigm followed by a review of most of the proposed algorithms for solving CSPs and DisCSPs for binary and non-binary constraints. Some of these techniques will be used in our experimental comparative evaluation.

Chapter 3

Local Consistencies for Constraint Networks

Constraint propagation (or filtering) techniques are in the core of constraint programming. They involve removing local inconsistencies from a CN. These techniques can be applied as a preprocessing step or throughout the search of solutions in order to encounter the major difficulties of search algorithms which is thrashing caused by local inconsistency [65]. Hence, a LC is a relaxation of consistency, which mean that for any CN *N* there is an equivalent non-empty locally consistent CN *N'*. *N'* is unique and can be found in polynomial time by so-called enforcing or filtering algorithms.

Local inconsistencies can be defined by single values or combinations of values that cannot belong to any solution because they violate some constraints. For example, assume that we have a value v_{1_k} for a variable X_1 and there is no possible value v_{2_l} for another variable X_2 that satisfies C_{12} (where X_1 and X_2 are related by a constraint C_{12}). Therefore the value *a* cannot belong to any solution because it does not satisfy the consistency between the two variable X_1 and X_2 and that we call, arc-consistency property[1]. Several levels of local consistency have been proposed in the literature. These levels will be given in details in Section3.1.

However, LC enforcement does not involve only values pruning. The transformation of the CN may involve (but not only) also the reduction of some constraint relations or may lead to the integration of new constraints to the CN [16]. Note however, in all cases the set of variables X should remain unchanged in order to ensure the equivalence property.

Notation 1 Given two constraint networks N (X, D, C) and N' (X', D', C'), we note N $\preceq$ N' if and only if X=X', C=C', and D(X) $\subseteq$ D'(X').

Definition 16 *Given a constraint network N, and a local consistency LC, The closure LC(N) is the constraint network N' such that N' $\preceq$ N and, for all local consistent constraint network N" such that N" $\preceq$ N, we have N" $\preceq$ N'.*

[1] Arc-consistency is one of the existing levels of consistency that will be given in more details below.

Definition 17 *Given two networks N and N'. N is equivalent to N' if and only if N and N' have the same set of solutions, sol(N)=sol(LC(N)).*

The author in [38] has defined a generic notion of consistency called (i, j)-$consistency$.

Definition 18 *A problem is (i, j)-$consistent$ if any solution of a subproblem including i variables can be extended to a solution including any additional j variables.*

In the following we will present some properties for some levels of local consistency followed by some of the existing enforcement techniques.

3.1 Properties of some levels of local consistency

In the following we will review some of the proposed LC properties for binary and non-binary CN. We give first some useful definitions of the notion of support for general CN.

Definition 19 *Let* P *(X, D, C) be a CSP and a constraint $C_{ij...} \in C$ and $X_k \in Var(C_{ij...})$. A value $v_{k_m} \in D(X_k)$ has a **support** in $C_{ij...}$ if and only if there is a tuple t that satisfies $C_{ij...}$ and such that $t[index(C_{ij...}, X_k)]=v_{k_m}$ [2]. t is then called the support of (X_k, v_{k_m}) in $C_{ij...}$.*

Definition 20 *For each variable X_i the neighborhood of X_i is the set of all the variables X_j adjacent to X_i in the constraint graph, i.e. every variable X_j involved with X_i in the same constraint.*

Definition 21 *A value v_{i_l} of a variable X_i, denoted as (X_i, v_{i_l}), is viable if and only if it has at least one support in the domain of every variable X_j in the neighborhood of X_i.*

The simplest local consistency is referred to as *node-consistency*.

Definition 22 *A CSP P (X, D, C) is **node-consistent** if and only if, for each $X_i \in X$, for all $v_{i_l} \in D(X_i)$; v_{i_l} satisfies all the unary constraints involving X_i.*

Example 2 Consider a variable X_i with $D(X_i)=\{-2, -1, 1, 2, 3\}$ and a unary constraint C_j: $X_i \geq 0$. Node-consistency enforcement technique will remove the values $\{-2, -1\}$ from $D(X_i)$.

3.1.1 Local consistencies for binary CN

Arc-consistency property

This level is the most used level of LC due to its low time and space complexities. Enforcing arc-consistency on a CN removes every value that has no support on at least one involved constraint. The deletion of a value may lead to the loss of support for another variable/value. Thus, the value deletions have to be propagated through the network since it can lead to value inconsistency of other values detected as viable previously. This process is known as *constraint propagation*.

[2] $index(C_{ij...}, X_k)$ returns the index of X_k in $C_{ij...}$

Definition 23 *A binary CSP is* ***arc consistent*** *[65] if and only if it has non-empty domains and each of its constraints is arc-consistent.*

Definition 24 *A binary constraint C_{ij} is arc-consistent if and only if each value v_{j_k} of each variable $X_j \in X(C_{ij})$, i.e. Var(C_{ij})=$\{X_i, X_j\}$; v_{j_k} has a support in X_i (vise versa).*

Example 3 Let's consider for example a CSP formed by three variables $\{X_1, X_2, X_3\}$ and three constraints relating these variables. The graph in Figure3.1 shows the allowed pairs of values. To make the domain of X_1 arc consistent, the value *c* need to be pruned because it has no support in the domain of X_3. However, the value *c* of X_2 will loose his support in X_1 and consequently the domain of X_2 will become arc-inconsistent. Due to the deletion of (X_1, *c*), (X_2, *c*) will be also deleted by constraint propagation. Figure3.2 shows the resulting problem after enforcing arc-consistency on all the variables' domains.

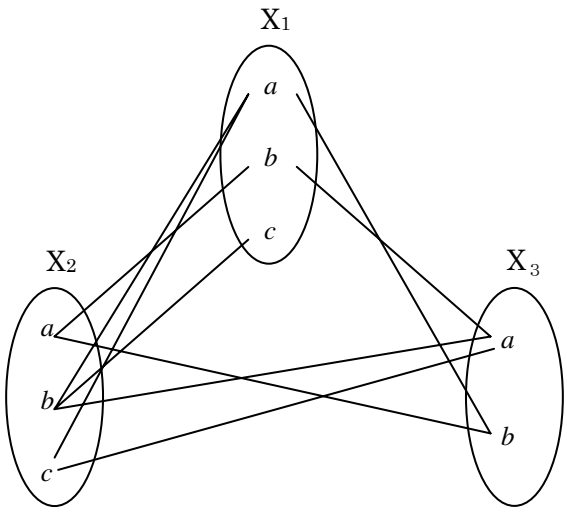


Figure 3.1. A graph based on possible consistent pairs of values of a constraint problem formed by three variables.

However, any consistent CSP is arc-consistent but the inverse is not always true. The example in Figure3.2 is arc-consistent but it is inconsistent problem, e.i., there is no solution that can satisfy the three constraints.

For binary constraint, [7, 8] proposed the property of bidirectionality of constraints. It is defined by the fact that for any binary constraint C_ij such that Var(C_{ij})=$\{X_i, X_j\}$:

- never checks (v_{i_l}, v_{j_k}) if there exists v_{j_f} still in D(X_j) such that (v_{i_l}, v_{j_f}) has already been successfully checked for C_{ij},
- never checks (v_{i_l}, v_{j_k}) if there exists v_{j_f} still in D(X_j) such that (v_{j_f}, v_{i_l}) has already been successfully checked for C_{ij},

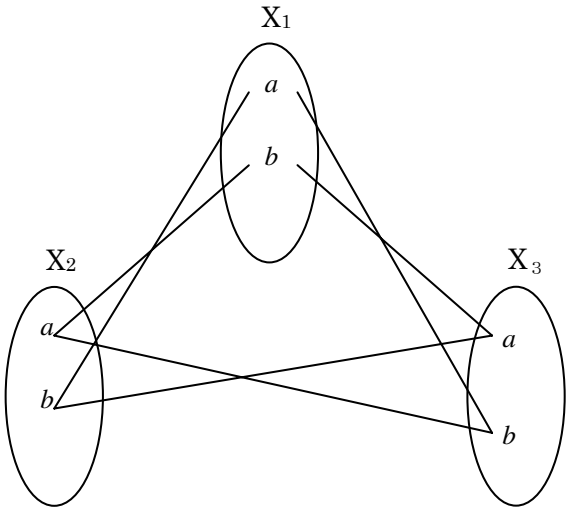


Figure 3.2. The resulting problem after enforcing arc-consistency.

This property is based on the use of constraint metaknowledge to infer or avoid constraint checks. Let's consider our previous example, while enforcing arc consistency on the domain of X_1 we found that (X_3, a) is the support of (X_1, b) and (X_3, b) is the support of (X_1, a), knowing that the constraint relating X_1 and X_3 is symmetric, thus no need to check the arc-consistency of the domain od X_3, by bidirectionality property this domain is arc-consistent.

Path-consistency property

Path consistency (PC) property is a higher level of LC that may delete more values than arc-consistency. Enforcing PC requires checking the path viability of each consistent pair of variables/values (X_i, v_{i_l}) and (X_j, v_{j_k}). This means, checking the existence of viable value for each variable along any path between X_i and X_j.

Definition 25 *A CN N is **path-consistent** [65] if and only if, all paths in P are path consistent.*

Definition 26 *A path Ch = $\{X_i, \ldots, X_h, \ldots, X_j\}$ in a CN N is path-consistent [17], if for all consistent pair of values for (X_i=v_{i_l}, X_j=v_{j_k}), i.e., (X_i=v_{i_l}, X_j=v_{j_k}) satisfies C_{ij}, one can find values for the intermediate variables X_h so that all the constraints $C_{il}, \ldots, C_{mh}, \ldots, C_{pj}$ in N along the path are satisfied (see Figure3.3).*

However, Montanari [67] showed that a CSP is path-consistent if and only if the completion of its constraint graph is PC. This latter means that in the complete graph, every path of length two is PC. Therefore, existing techniques for enforcing PC, proceed first by completing the sparse graph by adding universal binary constraints, then enforce PC on each path of length two. This process requires high computational complexity. Despite this cost, PC can

be used to find solution for binary convex CSP in a backtrack-free manner according to the following theorem [102].

Theorem 1 *Let N be a path consistent binary constraint network. If all the binary relations are row convex or can be made row convex, then the network is minimal and globally consistent.*

Knowing that a globally consistent network have the property that a solution can be found without backtracking.

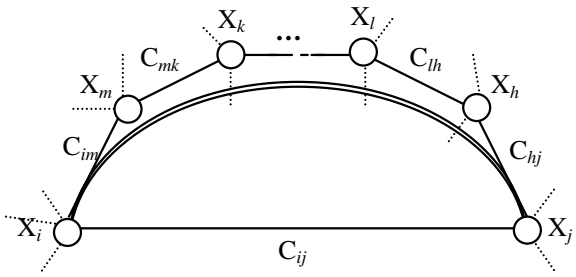


Figure 3.3. Path consistency.

When a path-consistent problem is also arc-consistent and node-consistent it is strongly *strongly* path consistent.

Example 4 Consider the example proposed by Stergiou in [91] formed by three variables X_1, X_2, and X_3 related by two all-different constraints C_{12} and C_{13}. The original example is arc-consistent (Figure3.4(a)) since each variable value has a support. However enforcing PC will induce a new equality binary constraint, C_{23} as in Figure3.4(b).

It is noteworthy that PC property has received particular interest in the area of temporal reasoning [99] where lower forms of consistency prove to be of less interest.

k-consistency property

Definition 27 *A CSP is **k-consistent** [38] if and only if, for all (k-1)-assignment (k-1) consistent can be extended to any additional k^{th} variable.*

The time and space complexities to enforce k-consistency are polynomial with the exponent depending on k. Nevertheless, for k $\geq$ 3, enforcing k-consistency requires the addition of new constraints which may change the structure of the CN. This leads to huge space requirements and subsequently to an important CPU time cost. The cost of the enforcing technique increases with the level to enforce, in practice, only arc-consistency can be used, while for path consistency it can be used only on small problems.

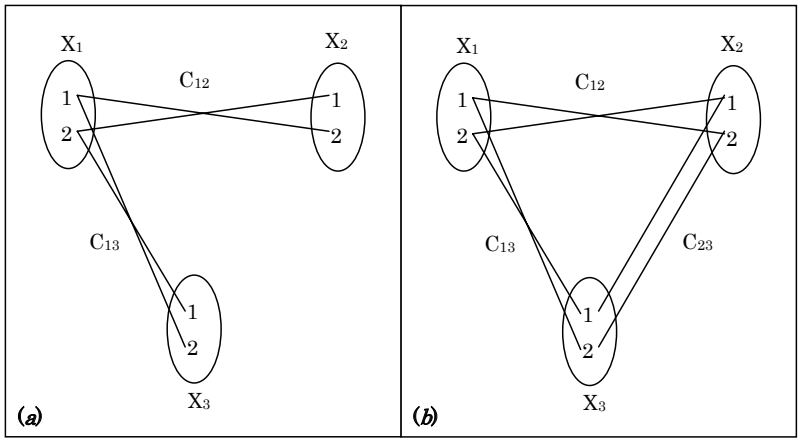


Figure 3.4. Example of an arc consistent problem for which we would enforce path consistency. The original problem (a) and the resulting path consistent problem with a new constraint structure (b).

Restricted path consistency property

Arc-consistency and path consistency properties were the most known levels of consistency. Ac is more used in practice than PC due to the drawbacks of the PC enforcement. Hence, Berlandier [5] proposed a partial level of consistency, restricted path consistency (RPC), in order to prune more values than AC while trying to avoid the drawbacks of PC. The basic of RPC is to perform only the most pruningful PC checks. In addition to AC, RPC checks whether pairs of values (v_{i_l}, v_{j_k}) of variables X_i and X_j respectively, such that v_{j_k} is the *only support* of v_{i_l} on C_{ij}, are path consistent. If the pair (v_{i_l}, v_{j_k}) is path-inconsistent, its deletion would lead to the arc-inconsistency of v_{i_l}. Thus v_{i_l} can be removed. These deletions make RPC able to prune more values than AC while it is cheaper than PC and without having to delete any pair of values, and so without changing the structure of the network.

Definition 28 *A binary CN is* ***Restricted Path Consistent*** *(RPC) [5] if and only if:*

- $\forall$ *$X_i \in X$, $D(X_i)$ is non empty arc consistent domain and,*
- $\forall$ $v_{i_l} \in D(X_i)$*, for all* $X_j \in X$ *such that a has a unique support* $v_{j_k} \in D(X_j)$*,*
- *for all* $X_k \in X$ *linked to both* X_i *and* X_j, $\exists$ $v_{k_m} \in D(X_k)$ *such that* (v_{i_l}, v_{k_m}) *satisfies* C_{ik} *AND* (v_{j_k}, v_{k_m}) *satisfies* C_{jk} ($C_{ik}(v_{i_l}, v_{k_m}) \wedge C_{jk}(v_{j_k}, v_{k_m})$).

Example 5 Consider a problem formed by three variables X_1, X_2, and X_3 all with domain $\{1, 2\}$ and three constraints, $X_1 \leq X_2$, $X_1 = X_2$, and $X_2 \neq X_3$ this problem is arc consistent (Figure 3.5(a)). The variable/value (X_1, 2) has only one support in X_2, which is (X_2, 2), but the pair (2, 2) of respectively X_1 and X_2 is path-inconsistent, i.e., this pair of values cannot be

extended to a consistent instantiation including the variable X_3. Therefore, enforcing RPC will remove the value 2 from the domain of X_1. Same process will be used to remove 1 from X_2. The resulting RPC problem is given in the Figure 3.5(b).

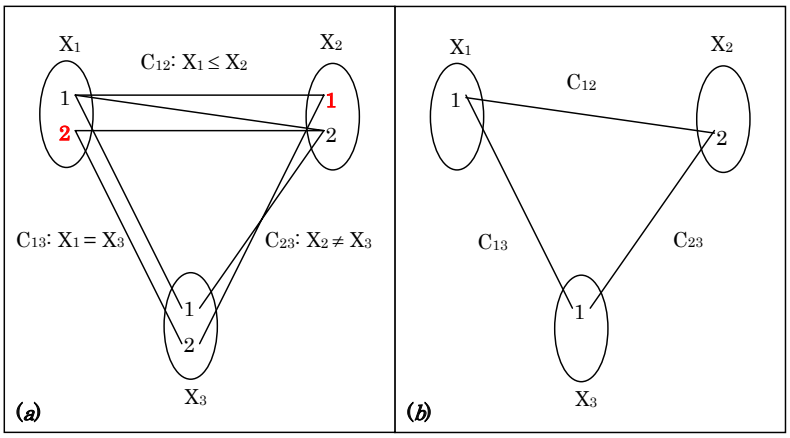


Figure 3.5. Example of arc-consistent problem for which we would enforce restricted path consistency. The arc-consistent original problem (a) and the resulting RPC problem (b).

Several extensions of RPC to more pruningful local consistency have been proposed in the literature. The authors in [32] extended the idea of RPC to deal with *k*-supports instead of *only one* support. The basic idea is that checking the path consistency of more supports may remove more values without falling in the drawbacks of PC. Their LC property, k-restricted path consistency looks for path consistent support on a constraint for each value having at most k supports on this constraint.

Another extension of RPC, max-restricted path consistency, was introduced by Debruyne and Bessière in [32]. A constraint network is max-restricted path consistency if all the values have at least one path consistent support on each constraint, whatever is the number of supports. Enforcing Max-RPC involves deleting all the *k*-restricted path inconsistent values for all k.

Other local consistency properties

Several other levels of LC have been proposed in the literature to prune more values than AC without falling into the traps of PC. Debruyne and Bessiere introduced in their work [31] the singleton consistency (SC) property. This property is based on the following remark: if a value v_{i_l} of a variable X_i is consistent, the CN obtained by restricting the domain of X_i to the singleton $\{v_{i_l}\}$ is consistent. Enforcing SC on a problem P consists of checking the inconsistency of the sub-problem $P|_{D_i=\{v_{i_l}\}}$. $P|_{D_i=\{v_{i_l}\}}$ denotes the obtained problem by restricting the domain of X_i to $\{v_{i_l}\}$. Many singleton consistencies can be considered,

amongst singleton arc-consistency [31]. To enforce SAC, we can apply any AC algorithm to check whether the sub-problem $P|_{D_i=\{v_{i_l}\}}$ is arc-inconsistent. As mentioned in [31] if the local consistency can be enforced in a polynomial time, the corresponding singleton consistency can be also has a polynomial worst case time complexity.

Freuder and Elfe have introduced another level of local consistency [39] to achieve high order local consistencies with good space complexity. The idea of the new level, inverse consistency (IC), is to remove values from variables that are not consistent with any consistent instantiation of some set of additional variables. Many inverse consistencies have been proposed. Generally, k-inverse consistency (or (1, k-1)-consistency according to Freuder [38]) removes the values that cannot be extended to a consistent instantiation involving any k-1 additional variables. The advantage of inverse consistencies techniques is that they only delete values from variables without adding new constraints and then save space. Nevertheless, k-inverse consistency can be applied only for small values of k due to the time complexity tat is polynomial with the exponent dependent on k.

The first level of k-inverse consistency is the path inverse consistency[3] (PIC) given in [39]. In [32] the authors show that a CP is PIC if and only if it is arc-consistent and for each variable/value (X_i, a), for any clique of three variables X_i, X_j and X_k, the assignment (X_i, a) can be extended to a consistent instantiation of X_i, X_j and X_k.

Another level where also introduced in [39] the neighborhood inverse consistency (NIC). This level checks the consistency of a variable/value (X_i, a) and its neighborhood.

amongst: singleton consistency and inverse consistency. A theoretical comparison of the existing levels of LC have been done in the literature, we will give an overview of the result of this result in Section 3.2.

3.1.2 Local consistencies for n-ary CN

Generalized arc-consistency property

Definition 29 *A non-binary CSP is a generalized arc-consistent (GAC) [72] if and only if for any variable in a constraint and value that it is assigned; there exist compatible value for all the other variables in the constraints.*

Example 6 Consider a non-binary constraint involving four variables $\{X_1, X_2, X_3, X_4\}$ with domains $\{1, 2\}$, $\{1, 4\}$, $\{1\}$, and $\{1, 2\}$ respectively. This constraint requires that the sum of the four variables is less or equal to 6. Hence, the value 4 of the variable X_2 has no tuple support in this constraint, i.e., there is no viable tuple including X_2=4. Enforcing GAC will remove the value 4 from the domain of X_2.

Definition 30 *For non-binary constraint, [14] proposed the property of* multidirectionality *of constraints. It is defined by the fact that for any constraint $C_{ij...}$, a tuple t on Var($C_{ij...}$)*

[3] Arc inverse consistency is equivalent to arc-consistency, (1, 1)-consistency.

is a support for the value t[index($C_{ij...}$, X_k)] where $X_k \in$ Var($C_{ij...}$) if and only if for all $X_h \in$ Var($C_{ij...}$), t is a support for t[index($C_{ij...}$, X_h)]. We say that an algorithm "deals with" multidirectionality if and only if

- *it never checks whether a tuple is a support for a value when it has already been checked for another value, and*
- *never looks for a support for a value on a constraint $C_{ij...}$ when a tuple supporting this value has already been checked.*

Other local consistency property

In [102], the authors have proposed another definition of arc-consistency for non-binary constraint network, namely *relational arc-consistency*. This definition requires global consistency on the subnetwork formed by the variables of the constraint and all the other smaller constraints implying some of these variables.

For some CN applying arc-consistency or stronger local consistency can be too expensive especially for large domain problems and for continuous domains where values are real numbers or floating point numbers. Bound consistency [61] (known also in the literature as *interval consistency*) is an approximation of arc-consistency which requires checking only lower and upper bounds of a variable domain. This property can be applied to any CN with any arity.

Definition 31 *A CSP (X, D, C) is **bound consistent** if and only if for all $X_i \in X$ is bound consistent. A Variable X_i is bound consistent if and only if $D(X_i) \neq \varnothing$ and min($D(X_i)$) and max($D(X_i)$) are consistent with each $C_{ij...}$ such that $X_i \in$ Var($C_{ij...}$).*

Example 7 *[82] Consider the CSP with six variables $X_1, \ldots, X_6$; with the following domains, $X_1 \in$ [3, 4], $X_2 \in$ [2, 4], $X_3 \in$ [3, 4], $X_4 \in$ [2, 5], $X_5 \in$ [3, 6], and $X_6 \in$ [1, 6]; and a single constraint alldifferent($X_1, \ldots, X_6$). Enforcing bounds consistency on the constraint reduces the domains of the variables as follows: $X_1 \in$ [3, 4], $X_2 \in$ [2], $X_3 \in$ [3, 4], $X_4 \in$ [5], $X_5 \in$ [6], and $X_6 \in$ [1].*

3.2 Theoretical comparison of local consistencies

In the previous section, we discussed several levels of local consistency that have been proposed in the literature. These levels can be compared in term of pruning efficiency of the corresponding enforcement techniques. However, among these levels, arc consistency and partial forms of arc consistency are the most used for their low space an time complexity. Higher levels are more costly but recently they became more useful for large problems.

Debruyne and Besiere in [33] proposed a qualitative study of the relations between various local consistencies. These relations are based on the transitivity relation "*stronger*" introduced in [31].

Definition 32 *A local consistency LC is **stronger** than another local consistency LC' if in any CN in which LC holds, LC' holds too.*

Hence, if a local consistency LC is stronger than another level LC' then any algorithm achieving LC deletes at least all the values removed by an algorithm achieving LC'.

Definition 33 *A local consistency LC is **strictly stronger** than another local consistency LC' if LC is stronger than LC' and there is at least one CN in which LC' holds and LC does not.*

Figure 3.6 taken from [33], summarizes the study performed by Debruyne and Bessière in [33] of the relations between some of the levels of LC mentioned above for binary CN according to their pruning efficiency.

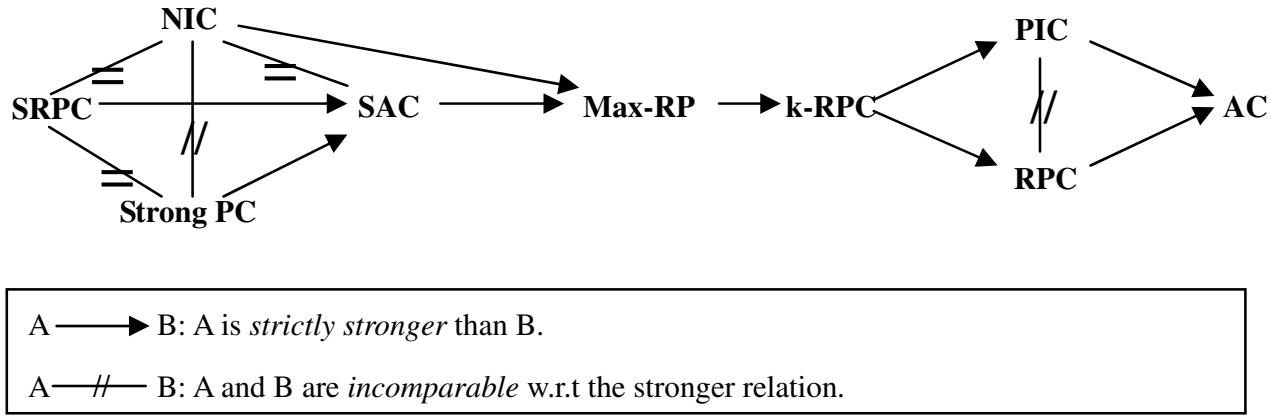


Figure 3.6. Relations between some levels of local consistencies for binary CN.

3.3 Local consistency enforcement techniques

Filtering techniques (known also as constraint propagation techniques) can be used to detect the inconsistency in a CN, and under some assumption they can ensure a backtrack-free search [37]. The main purpose of these techniques is not to find a solution in a CN, but to remove some local inconsistency and hence detect some regions in the search space that do not contain any solution. However, applying these techniques does not guarantee that all the remaining values are parts of solutions. The main gain behind these techniques is that in case of a substantial reductions are made the search becomes easier. These techniques can be divided into two main groups, centralized techniques and distributed techniques.

In this section we review some of the proposed efforts on constraint propagations. We focus essentially on arc consistency techniques for both centralized and distributed framework and for binary and non-binary CNs, since they play important role in our research. For other techniques we will just give a general overview because they are outside the main scope of this thesis.

3.3.1 Centralized techniques

For binary constraint networks

The first level of local consistency existing in the literature is node consistency (NC). This level is established by NC-1 algorithm [65]. This algorithm performs a consistency check for each node X_i in the CN. However, for a CSP *P* with *n* variables, *d* size of largest domain, the NC enforcement can be be performed in O(*nd*).

As for AC enforcement, it can be obtained for any binary CSP P while using the following domain restriction operation. Assume that *G* is the constraint graph associated to *P*, *Links*(*G*) denotes the set of possible links in the constraint graph *G*.

$\forall (X_i, X_j) \in Links(G)$: $D(X_i)=\{v_{i_l} / v_{i_l} \in D(X_i), v_{j_k} \in D(X_j)$ and (v_{i_l}, v_{j_k}) satisfies $C_{ij}\}$.

Hence, we have to check all the links (X_i, X_j) in *G* and remove each value $v_{i_l} \in D(X_i)$ (*resp.* $v_{j_k} \in D(X_j)$) that has no support in $D(X_j)$ (*resp.* $D(X_i)$).

Most research efforts were devoted to arc-consistency enforcement due to its low cost for both binary and non-binary problems. Thus, there has been a number of proposed algorithms in the literature. These algorithms can be divided into three main groups according to the type of schema used, i.e., type of information propagated, in case of constraint propagation.

- Constraint oriented propagation schema (AC-1, AC-2 and AC-3 [65], AC2000, AC-2001 [9]),
- Variable oriented propagation schema (AC-3 [64], AC2000, AC-2001 [9]),
- Value oriented propagation schema (AC-4 [71], AC-6, AC-Inference and AC-7 [7, 8])

The first algorithm proposed in the literature for enforcing arc-consistency on any binary CN is AC-1 [65]. The basic of AC-1 relies on a systematic revisions of all the links in the constraint graph in case of a value deletion. However, the deletion of a value may have direct impact only on the neighborhood variables. The temporal complexity of AC-1 is $O(n^3d^3)$ while its spacial complexity negligible, i.e., no extra data structure is used by AC-1.

As a remedy to the limitation of AC-1, Waltz proposed another algorithm AC-2 [104] based on AC-1. The basic of AC-2 is that in case a revision of a link (X_i, X_j) yield to the deletion of a value v_{i_l} from $D(_i)$ revise all the links (X_i, X_k) where $k < i$ and $k \neq j$. AC-2 uses two queue structures to store the links needed to be revised at eat iteration.

AC-3 is another extension of AC-1, that uses the same property as AC-2. This algorithm was discussed in [65] uses only one queue structure. The temporal complexity of this algorithm is $O(n^2d^3)$ while its spacial complexity is $O(n^2)$.

Mohr and Henderson proposed another arc-consistency algorithm AC-4 [71] where the revision of a variable's domain is based only on the values supported by the deleted one. In other words, if a value *a* is deleted from a domain of a variable X_i, then this deletion affects directly only the values of neighborhood variables supporting *a*. AC-4 seeks first all

supports for each variable value in the constraint graph which may increase the complexity of the this algorithm especially when the number of allowed pair of values is high. AC-4 computes a total support count initially and then updates it as values are deleted. The spacial and temporal complexities of AC-4 are O(n^2d^2).

For some period the state of the art resided in two algorithms, AC-4 for its optimal worst-case behavior and AC-3 which often exhibits better average-case behavior. Two other algorithms, AC-5, proposed in [30] and in [58], and another in [79], permit exploitation of certain specific constraint structures, but reduce to AC-3 and AC-4 in the general case.

Bessiere and Cordier developed AC-6 [6], which retains the optimal worst-case behavior of AC-4 while improving the average-case behavior of AC-3. AC-Inference [8] owes something to all these predecessors, but permits the use of inferred support; while AC-7 [7, 8] is the most closely related to AC-6. AC-7 is an hybrid of AC-4 and AC-6 while using the bidirectionality property of relations associated to constraints.

The basic idea of the AC-7 algorithm consists on two main operations: seeking a current support for a value, and processing the deletion of a value. For each value, AC-7 seeks a support in each related constraint. It introduces a total ordering between values in each domain, it computes one support (the first one) for each label (X_i, v_{i_l}) on each constraint C_{ij} if and only if this support can not be inferred by bidirectionnality. In fact, AC7 never searches a support $v_{j_k} \in D_j$ for a value $v_{i_l} \in D_i$, according to the constraint C_{ij}, if there exist a value v_{j_f} still $\in D_j$ and v_{i_l} has been already successfully checked as a support of v_{j_f}. Therefore AC-7 could save d^2 constraint checks. The total space complexity of AC-7 is *O(ed)* while its time complexity is *O(ed^2)*, with *d* is the size of the largest initial domain and *e* the number of constraints of the CSP. The AC-7 algorithm will be used in our experiment, with the DRAC approach as a witness approach to evaluate the final result and to ensure the efficiency of our proposed DRAC.

Two refinements of AC-3 (known as the simplest algorithm for its data structure), AC2000 and AC2001 [7] are suggested in the literature for binary problems. AC-2000 is a refinement of AC-3 while avoiding blind search for new support for each value *a* in D(X_j) in case of a domain reduction in D(X_i) (X_i and X_j are neighborhood). In case AC-2000 uses a variable oriented propagation schema, its spacial complexity is O(nd). This complexity is O(ed) in case of constraint oriented propagation schema, where e is the number of constraints in the CN. AC-2001 is a refinement of AC-2000 in which the authors saved more constraint checks. The spacial complexity of AC-2001 is O(ed) while its temporal complexity is O(ed^2).

Concerning higher consistency, as indicated earlier, few centralized works were directed toward *k*-consistency with $k > 2$ in the literature due to its huge cost. The best centralized algorithms proposed for reinforcing PC (3-consistency) are PC-5 [33], with O(n^3d^3) worst-case time complexity and O(n^3d^2) worst-case space complexity, and PC-8 [23], with O(n^2d)[4] worst-case time complexity and O(n^3d^4) worst-case space complexity.

[4] As mentioned in [31] this algorithm still requires O(n^2d^2) data structure for the constraints representation.

For restricted path consistency (RPC), the underlying centralized proposed technique removes more inconsistencies than AC while avoiding the drawbacks of PC. The RPC-1 algorithm described in [5] is based on the principle of AC-4 and has O(*ed(n+d)*) worst-case time and space complexity where *e* is the number of constraints.. In [32] the authors proposed another RPC algorithm, RPC-2, based on the principle of AC-6. The space complexity of this algorithm is O(*end*) and its temporal complexity is O(end^2) in the worst-case. Max-RPC [32] is another extension for RPC-2 with O(end^3) temporal complexity and O(end) spacial complexity.

For non-binary constraint networks

As for GAC-7 [11], the general schema to general constraint networks, it is based on AC-7 algorithm and able to efficiently handle any constraint of the industrial applications. It makes the use of "current support" idea, and of "multidirectionality" (the generalization of bidirectionality to non-binary constraints) in order to save as many constraint checks as possible. This algorithm never checks whether a tuple is a support for a value when it has already been checked for another value, and never looks for a support for a value on a constraint C when a tuple supporting this value has already been checked. Thus GAC-7 can save d^r constraint checks on *r*-ary. constraint. The authors propose many frameworks for searching supports depending on the type of the constraint.

The total space complexity of GAC-7 is O(r^2d) while its time complexity is *O(ed^r)*.

3.3.2 Parallel and distributed techniques

As mentioned in [3], concerning parallel arc-consistency algorithms, the first algorithms were developed for the shared memory paradigm: Waltz in [104] designed parallel versions of AC1, AC3 and AC4 for shared memory computers. They have used *p* processors with the complexity O($\frac{n^2d^2}{p}$) with *n* is the number of variables, *d* is the size of the largest domain. In [21] the authors implemented and experimented a parallel version of AC-4 for the connection machine CM-2 with a complexity of O(*ndlog(nd)*) due to the communication overheads.

As for distributed algorithms, Nguyen and Deville, in [78], proposed DisAC-4 algorithm, a coarse-grained parallel algorithm designed on the basis of AC-4 and the DisCSP formalism, which defines an agent as responsible of a subset of variables. DisAC4 is used for a distributed memory computer using asynchronous message passing communication. Unfortunately, it has been restricted to diffusion networks (*Ethernet*), which leads to an underlying synchronism between processes. The theoretical complexity is O($\frac{n^2d^2}{k}$), where *k* is the number of the processors.

In [53] the author proposed DisAC-6, it is based on AC-6 and DisCSP formalism. The basic idea of this algorithm is to scatter the problem among autonomous processes and make them asynchronously interact by point-to-point messages containing useful information (in

order to perform the global arc-consistency). The worst time complexity is O(n^2d^3) and the space complexity is O(n^2d) with O(*nd*) the amount of message operations. DisAC-9 is an improvement of DisAC6. It is an optimal algorithm in the number of message passing operations. It exploits the bidirectionality property of constraint relations, which allows agents to induce acquaintances relations. The worst time complexity of this algorithm is O(n^2d^3) with *nd* messages and with a total amount of space in O(n^2d).

Note that the goal of DisAC-9 is essentially to reduce the total amount of messages by doing more local computations, because of the high cost of messages passing in a distributed multiprocessor architecture. As we intend to use a mono-processor machine, we ignore the cost of messages passing, and rather focus on reducing the local agent computation.

Table3.1 summarizes the worst case time and space complexities of the most existing efficient algorithms for achieving different levels of local consistencies.

Table 3.1. The temporal and spacial complexities for the most efficient existing algorithms for enforcing different levels of local consistencies with n, the number of variables, d, the size of the initial largest domain, e the number of constraints, c the number of 3-cliques in the graph, and r the arity of the constraints.

Algorithm	Time complexity	Space complexity
AC-3	O(ed^3)	O($e + nd$)
AC-4	O(ed^2)	O(ed^2)
AC-6	O(ed^2)	O(ed)
AC-7	O(ed^2)	O(ed)
AC-2000	O(ed^3)	O(ed)
AC-2001	O(ed^2)	O(ed)
RPC2	O($en + ed^2 + cd^2$)	O($en + cd$)
Max-RPC	O($en + ed^2 + cd^3$)	O($en + cd$)
PC-5	O(n^3d^3)	O(n^3d^2)
PC-8	O(n^3d^4)	O(n^2d)
GAC-4	O(ed^r)	O($ed^r + nd$)
GAC-7	O(ed^r)	O(er^2d)
DisAC-6	O(n^2d^3)	O(n^2d)
DisAC-9	O(n^2d^3)	O(n^2d)

3.4 Summary

In this chapter we presented an overview of local consistency property. We reviewed first various levels of local consistency for binary and non-binary CN. Then, we gave theoret-

ical comparison of some local consistencies. Finally, we described some of the proposed techniques for enforcing local consistency on binary and non-binary CN. Some of these techniques will be used in our experimental comparative evaluation.

Chapter 4

Meeting Scheduling Problem

In this chapter we define first the meeting scheduling problem (MS) as well as its main features. Second we will present the Clarke Tax (CT) mechanism proposed in the literature to enhance the efficiency of a multi-part cooperative solver where the quality of the result depends especially on the quality and how trustworthy and reliable the received information is. Then we discuss some of the proposed research efforts deling with MS problem followed by an illustration of an important arises issue, privacy issue. Finally we summarize this chapter.

4.1 Definition

In our daily life, meeting scheduling (MS) is a pre-eminent and typical group decision support problem that embodies a decision-making process affecting several users. Each user is assumed to be self-interested. That is every user has its own preferences and desires about how the world should be and often makes his decisions based on them. The preference over a set of alternatives Ω can be measured by means of *utility functions*. A utility function u_i:$\Omega \rightarrow \mathcal{R}$ assign to every alternative a real number indicating how "good" the alternative is for the user. The larger the number the better from the point of view of the agent with the utility function [105].

A MS problem can be described by the process of scheduling events (meetings) involving individual constraints, i.e., private preferences over alternative mutual decisions. These constraints are crucially related to the availabilities and preferences of the users who should participate in the meetings. Solving MS problem involves determining *when* and *where* one or more meeting(s) should be scheduled depending on the available times, timetabling, and preferences of the involved users. This task is normally time-consuming, iterative, and sometimes tedious.

The two main features of the MS problem are defined respectively by its naturally distribution and its dynamic environment. For the first, the participants in the MS problem may

not belong all to the same organization or even to the same city, hence this problem cannot be solved by a centralized approach. As for the second features, two kinds of MS problem can be defined static and dynamic problems leading respectively to two types of MS schedulers static and dynamic scheduler based on the definition of static vs. dynamic scheduling given in [85].

Definition 34 *A MS scheduler is called static (or pre-run-time) if it makes its scheduling decisions off-line and generates a complete schedule of the possible meetings at compile time. for this purpose it needs complete prior knowledge about all the meetings and the underlying participants. These information is needs at run time to decide at every point of a discrete time base which meeting is to be scheduled next.*

Definition 35 *A MS scheduler is called dynamic (or on-line) if it makes its scheduling decisions at run-time on the basis of available request for meetings. Dynamic MS scheduler is flexible to adapt to an evolving meetings scenario and have to be incremental, i.e., to reply to the new requests, adding or cancelation of meeting, without proceeding from scratch.*

Nevertheless, in dynamic environment users are frequently adding new meetings or removing scheduled ones from their calendar. This process often leads to a series of changes that must be continuously monitored. Hence, we need to find reach a compromise between all the attendants' meeting requirements[1] (i.e., date, time and duration) which are usually conflicting. Automating meeting scheduling is important, not only because it can save human time and effort, but also because it can lead to more efficient and satisfying schedules within organizations [40].

Moreover, in real organization or company, meetings do not have same degree of importance (same priority). Obviously, the great significance of a meeting depends especially, but not only, on the leader of the event, the number of participants, and the meeting's main topic. Therefore, the search should be for the optimal solution (satisfying some predefined optimality criteria such that maximizing the summation of utility function of all the involved users) whenever possible. Different kinds of optimality criteria have been suggested in game theory, economics and voting theory. Thus, the solver process should seek for a compromise among different human user's requirements regarding both the potential meetings' time and the meetings' priorities.

In addition, a multi-part problem requires a negotiation process among all the participant. However, the main concern in every negotiation protocol is usually that the agreed-upon decision will be optimal in some sense. The optimality is measure with respect to the participants' private preferences. The key question is

[1] In the sequel of the thesis, we use the term date to define the date, time and duration of a meeting, while for the place, we assume that all the attendants belongs to the same city and thus to simplify the problem.

Clarke Tax mechanism is a well known mechanism for revealing agents' preferences [34]. This mechanism, used in our proposed approach, is given in detail in the next section.

Recently another important issue is addressed by many researchers, the privacy issue.

4.2 Clarke Tax mechanism for ensuring truthful preferences

As mentioned in [34]. The basic idea of the clarke Tax mechanism (CT) is to incite each user to tell the *truth*. This mechanism is based on the sealed-bid mechanism, the most straightforward procedure to choose one alternative among many others. Each user bids on all the alternatives and the alternative that has the maximal sum of bids is chosen.

The basic idea of CT is not only to choose the alternative with highest bidding, but also to fine each user with a tax. The tax is computed according to the proportion of the user's bid that makes a difference in the outcome. To illustrate more clearly the idea of TC, let's consider the following example formed by four users and three alternatives. Assume that the users are asked to express their degree preferences using numbers from 1 to 30. Table 4.1 shows the truth preferences given by the three users. Table 4.2 shows the summation of preferences for each alternative without a_i preferences and the corresponding computed CT.

Table 4.1. Example of truth users' preferences for each alternative.

	a_1	a_2	a_3
user$_1$	20	33	14
user$_2$	8	30	17
user$_3$	16	2	28
user$_4$	23	12	5
Summ preferences	67	**77**	64

According to Table 4.1, the best alternative with maximum preferences is the alternative a_2. For each user *user*$_i$, we compute the summation of the preferences of all the other users *user*$_j$ ($i \neq j$) for each alternative a_i. For example if the user *user*$_2$ did not reveal his preferences, the winning alternative would be a_1 rather than a_2. The alternative a_1 would overtake a_2 with 12, i.e., 59-47=12. The bidding of *user*$_2$ has affected the result with a "magnitude" of 12. The tax computed for *user*$_2$ is 12. The Users *user*$_3$ and *user*$_4$ are not fined because their revealed preferences did not change the result, i.e., the same alternative always win with or without their bidding. Therefore the dominant strategy for the user$_2$ is to divulge his true preferences, otherwise he will pay more tax or will not get his choice.

Thus, according to Ephrati and Rosenschein in [34], higher the preference given by a user for an alternative higher would be the tax to pay in case he affects the result. The user who

Table 4.2. The Clarke Tax computed for each users.

	a_1	a_2	a_3	Tax
$user_1$	47	44	50	6
$user_2$	59	47	47	12
$user_3$	51	75	36	0
$user_4$	44	65	59	0

overestimate his preferences (to force winning for some preferred alternatives) risks having to pay tax more than his true preferences. Similarly, if the user underestimate his preferences (to save tax), the lost of his utility might be larger than the saved tax. The best strategy is then to reveal the truth preferences.

This mechanism can be used in our proposed approach to solve MS problems. However, the user in CT mechanism is provided periodically by a certain amount of points that he can use to estimate his preferences. The decision of the user on one stage may have impact on several forthcoming stages. In this case, the user may always underestimate his preferences for a set of *m* meetings in order to accumulate points and use them for his most interesting meeting *m*+1. Therefore, we propose to afford to each user for each meeting a fixed amount of points. The user has to split up the whole amount (minus the tax computed on the previous meeting) among all the possible dates. In this case The user cannot accumulate points. Hence, if the user will overestimate his preferences for a meeting m_1, then he has to pay high tax for the next meeting, and he may not have enough points to express his preferences. If the user will underestimate his true preferences, he will not get extra points for the next meetings.

In chapters 7 and , we will discuss the possibility of applying such mechanism to ensure truthfulness.

4.3 Basic of some meeting scheduling solvers

Many research efforts dealing with solving MS problem were proposed in the literature; among them there are those based on CSP (constraint satisfaction problem) formalism [67]. The underlying problem is formalized as centralized CSP in which all the users' information is centralized in the same process [1, 4]. These works are essentially focused on over-constraint CSPs.

However, recently multi-agent systems (MAS) are widely used to address many real-world combinatorial applications. Hence, recent researches have argued about how to solve MS problems using an agent-based approach for many reasons. The main reason is that agents can accomplish their tasks through cooperation while allowing the users to keep their privacies. A mechanism design approach based on multi-agent system (MAS) to solve MS problems was reported first by Ephrati et al. [34]. The authors defined two paradigms of

MS scenarios, open scheduling systems and closed scheduling system. The first system concerns cases where the users are independent, completely in control of their time resources, and have no obligation to meet each other unless it serves their own selfish interest. Hence, the users themselves determine the feasibility of each meeting. Whilst the second system, closed system such as company or organization, meetings are imposed on involved users. Thus every participant in a meeting have an obligation to attend the meeting, if feasible. The constraints are defined by the scheduling system and not by the participants. Therefore, the scheduling system maintains a consistent and complete global calendar of the organization's members.

The authors proposed three scheduling mechanisms, for the closed system, which differ in the information type that each user has to reveal about his individual preferences. The authors tried to approximate the optimal utilitarian choice while avoiding manipulability by using Clark Tax mechanism [35].

Garrido and Sycara [49] reported another MAS work that focused on using distributed autonomous and independent agents to solve the problem. Each agent has its individual goal, to schedule the meeting while maximizing its individual preferences. This work is based on the communication protocol presented in [96] where agents are capable of negotiating and relaxing their constraints in order to reach an agreement on a schedule with high join utility.

Sen et al. [92] have proposed another work based on how an application domain for intelligent surrogate agents can be analyzed, understood and represented in order to make these agents able to carry out tasks on behalf of human users, taking into account their environment. Their prior work has spotlighted on agents adapting to environmental changes [93], however, in [92] their efforts were directed towards the integration of user preferences. Often users' preferences are mutually conflicting, so the authors used techniques from voting theory to formally represent and reason with conflicting preferences.

Three other multi-agent approaches to MS problems, using the Partial CSP formalism introduced by [36], were given in the literature. The first work proposed by [63] offered a new approach for MS problems using fuzzy constraints. The underlying protocol is called the selfish protocol, where each user tries to maximize their preferences during the negotiation process.

The second in [101], used the distributed valued constraint satisfaction problem (DVCSP) formalism to model the MS problem. The authors propose to convert each already registered event into a constraint. A weight, an integer between 0 and 9, is assigned to each constraint and to each event to reflect its importance. Two kinds of agents are used in this model a group agent and a personnel agent. Each personnel agent is associated to a human user and acts on his behalf. As for the group agent is needed in order to maintain and to facilitate the scheduling process within a group. For each meeting, the proposer agent will communicate the necessary information to the group agent. The later agent will generate the needed possible times and send them the participants (within same group or from another group) with a

weight threshold *Tr*. The participant agents will reply by sending their personal constraints having weight greater than or equal to *Tr*. The group agent will search for a valid assignment according to the received constraints. If such assignment is found, then the group agent will broadcast it to all participants. Otherwise, a failure message will be broadcasted to all participants. In this protocol, the agents should reveal their constraints, preferences and calendar to the group agent. Even if this agent is considered a trusted party in the system, users prefer not to send any of their private information to another agent in the system. Therefore this protocol does not guarantee any level of privacy. In addition, to reach an agreement between participants, requires a high amount of messages of large sizes. This approach is used in our experimental evaluation.

The third work based on multi-agent systems and using fuzzy constraints to express users' preferences was presented by Franzin et al. [42]. Their meeting scheduling system was based on an existing system that includes hard constraints [41]. The authors proposed, in their work, to integrate preferences to their system and focused on observing the behavior of this new system under several conditions [41]. Their main objective is to evaluate the relations among solution quality, efficiency and privacy. The correlated protocol is based on inferring some new knowledge during the solving process which, may clash with the desires of agents to keep their information private. The authors propose a simple communication model which consists of several proposal phases for each meeting according to the expected result, the first feasible solution or the optimal solution. At each phase, a proposal is made by one of the agent and communicated to the others. This proposal is chosen among the best in the calendar of the proposer agent and checked as consistent with the knowledge collected about the other agents. The other agents which receives the proposal reply with their level of preferences, i.e., preferences=0 if the proposal is rejected, preferences $\neq 0$ otherwise. However, the knowledge about other agents (and also about the proposer) is updated according to the received proposal and the answer.

In case of a rejection, a new proposal phase is started otherwise, a solution is found with a preference value the minimum among all the received. In case of search for optimal solution, another proposal phase will start for the same meeting, except that all the preferences values which are smaller than or equal to the value of the last solution found are set to 0. this implies that a new negotiation phase will start to find better solution or solution with preference = 0. The author introduce in their system the concept of threshold to choose a feasible solution which is optimal in some sense. The main goal is to reduce the number of proposals and thus speed up the whole process.

In [42] the author suggested two basic global criteria to optimize for each agent, the fuzzy optimality; which consists of having preferences between 0 and 1 of maximizing the minimum preference across all agents, and the Pareto optimality [40], where a solution is optimal if there s no way to improve the preference of any agent without decreasing the preference of some other agents. However, in this protocol the number of exchanged messages increases

with the size of the problem (number of possible proposals and users). In addition, in the worst-case each agent has to reveal all its proposals in order to reach optimality.

Another research dealing with MS problems were contacted by Maheswaran et al., [68] at the same time as ours. The authors raised two important points in their work. The first one is the non-existence of a automated congruent mapping to distributed constraint optimization (DCOP) formulations. Their main motivation is that this mapping is a tedious process of modeling an environment, choosing variable sets, and designing constraint utility functions. The second point is that it is unclear if DCOPs obtained from concrete problems will fall within a space where complete algorithms for problems with *NP* complexity are fast enough to be utilized.

The authors have considered the DiMES framework (distributed multi-event scheduling) because it captures a rich class of real-world problems where multiple agents must generate a coordinated schedule for execution of joint activities or resource usage in service of multiple events. Their main idea is to convert a given DiMES problem into DCOP with binary constraints and then apply an existing (or improved) algorithms developed for DCOP to obtain an optimal solution. Three DCOP formulations were proposed, which differ in the used concepts for creating variable sets: time slots as variables, events as variables, and private evens as variables. They proposed for each variable set a constraint utility functions and they proved that the obtained solution from the DCOP formulation is congruent to the original DiMES problem.

Nevertheless, the majority of these works share the following properties:

1. Dealing only with non-dynamic problems (among which [1, 4, 101, 42],).

2. Allowing the relaxation of any user's preferences, even those related to non-availability of this user in order to arrive at consensus choices for a meeting's time. However in real-world applications this is not always permitted. For example, when the user is traveling on business, such a constraint would oblige the user to stop his/her travel to attend the meeting, and this is not always possible (amongst [96, 49, 92, 63, 101, 42]).

3. Not integrating the enforcement of local consistency in their solving process, in spite of the pre-eminent role of the filtering techniques in the efficiency of solving an NP-complete problem. Only the authors in [41, 42] deal with the use of some inferred knowledge to maintain coherence between meetings in order to steer the selection of the next proposal, while, none of the other works try to maintain any level of consistency during the negotiation process.

4. Judging all the meetings of the whole system with the same level of importance (among others [49, 63, 42, 101]). In real life, this is not always true. Obviously, the great significance of a meeting depends especially, but not only, on the leader of the event, the number of participants, and the meeting's main subject. Especially in a dynamic envi-

ronment, such discrimination may lead to conflicting meetings, and may also increase the number of meetings to reschedule.

5. Not considering the high complexity of message passing operations in real distributed systems ([49, 92, 63, 101, 42]).

4.4 Privacy issues

Privacy is a critical issue that usually arises while talking with cooperative communication involving independent agents endowed with information about their users. The assumption is often made that any requisite information will be shared. Agents may want to maintain and to protect as much as possible their individual users' privacy while engaging in collaborative problem solving. Therefore, in such system, it is often desirable to exchange as little information as possible in order to keep private the own data of the agents. Nevertheless, the main problem in exchanging information is that an agent can build an approximation of the private knowledge about other agents by accumulating information. However, the main question that arise is how to meet the added requirement of privacy maintenance while trying to solve problems efficiently.

However as we mentioned before, the quality of the solution would depend in a great part on the exchanged information. Recently, many research efforts were directed toward how to keep the privacy of users while searching for a solution to the problem.

Franzin et al. [41] have proposed an empirical study of the relations among the three main features of a multi-part cooperative system, privacy loss, efficiency and solution quality. Their obtained results show that the number of initial meeting and the threshold influence the level of such measures. In addition, the authors show that the search for a feasible solution is not slowed down by the addition of the preferences.

Earlier work on privacy focused on creating secure coordination mechanisms such that negotiation would not be observable to parties outside the collaborating set of agents [108].

Maheswaran et al., proposed a quantitative approach to deriving metrics for privacy for general domains [69]. Therefore they proposed a Valuation of Possible States (VPS) to quantitatively evaluate privacy loss in multi-agent settings. The authors applied their ideas in a distributed meeting scheduling domain modeled as a distributed constraint optimization problem (DCOP). The authors modeled the private information of a user as a state among a set of possible states. Hence, each user has an estimate of the likelihood that another user lies in in each of the possible states. Therefore they proposed to interpret privacy as on the other agents' estimates about the possible states that one lives on. The main objective of the authors is to build a unifying framework for privacy, in order to capture existing notions of privacy. The authors applied VPS to a personal assistant domain: distributed meeting scheduling

4.5 Summary

In this chapter we defined a real-world problem, meeting scheduling problem and its main features. We presented some of the encountered difficulties with this problems followed by some of the research efforts discussed in the literature.

Chapter 5

DRAC and GDRAC: Distributed Reinforcement of Arc Consistency for any General Constraint Network

In this chapter, we begin our investigation on distributed local consistency enforcement for binary CN. Hence, the main objective of this chapter is to propose two novel approaches for enforcing arc consistency on any CN in a totally distributed manner. We present first the DRAC approach (for Distributed Reinforcement of Arc consistency) followed by its generalization, G-DRAC approach, for any constraint's arity (general AC).

In the following we present first the multi-agent architecture followed by an illustration of the proposed heuristics. Then we describe the global constraint-agent interactions of the two proposed approaches DRAC and G-DRAC. Then, we give the proof of correctness and termination properties followed by a computation of the complexity of both approaches. Finally, we exhibit the experimental results and summarize this chapter.

5.1 Underlying multi-agent architecture

Multi-agent systems (MAS) are considered as natural metaphor for understanding and building a wide range of distributed real-world applications [105]. These systems are composed of multiple interacting computing elements, known as agents. Agents are computer systems with mainly two important capabilities: capable of autonomous action (to some extent), and capable of interacting with other agents. We have used these systems with constraint-based graph to model any distributed constraint satisfaction problem.

Hence our proposed multi-agent model, for both approaches, involves two kinds of agents (Constraint agents and Interface agent) in cooperation. The Interface agent, is an intermediate agent between the human user and the machine. it has been added in order to detect whether the full global arc-consistency has been achieved and, especially, to inform the user of the result.

Each agent has a simple structure: acquaintances (the agents that it knows), a local memory composed of its static and dynamic knowledge, a *mailBox* where it stores the received messages and a behavior.

5.1.1 Interface agent

The Interface agent has as acquaintances all the Constraint agents of the system. Its acquaintances, denoted by Γ, represent its static knowledge. The dynamic knowledge of the Interface agent consists of the internal state of all the agent constraints in the system.

5.1.2 Constraint agents

Each agent A_i has its own variables[1]. For simplicity reasons, assume that each agent A_i maintains only one constraints, denoted by $C_{ij...}^{A_i}$. However, both approaches can deal also with cases where each agent is responsible of a set of constraints.

As for the agent A_i acquaintances, they consist of both all the agents with which it shares at least one variable Γ^{A_i}, i.e., Γ^{A_i}=$\{A_j$ / $A_j \in \Gamma$ and $Var(C_{ij...}^{A_i}) \cap Var(C_{ij...}^{A_j}) \neq \emptyset\}$, and the Interface agent. Its acquaintances and its associated relation define its static knowledge. While its dynamic knowledge concerns its internal state, the domains of its own variables and a parameter called *EndBehavior*, which specifies whether its behavior is completed or not.

Two Constraint agents are connected together if and only if they share at least one variable. These links are known as inter-agent constraints. All the Constraint agents will negotiate and cooperate together to enforce arc consistency on the underlying problem. Therefore we assume the following communication model between all agents:

- The agents in the system negotiate by exchanging asynchronous point-to-point messages containing the necessary relevant information in a manner that reduces the number of messages passing.
- An agent can send a message to another one only if it knows that this agent belongs to its acquaintances.
- The messages are received in a finite delivery time and in the same order that they are sent. Messages sent from different agents to a single agent may be received in any order.

Note that the goal of our approach is to obtain the full global arc-consistency as a result of the interactions between the Constraint Agents by exchanging inconsistent values. In other words, the full global arc-consistency is obtained as a side effect of the interactions between reactive agents; each one of them having a local goal, i.e., to enforce arc consistency locally on the subproblem formed by A_i and its acquaintances Γ^{A_i}.

[1]The variables implied in the constraint maintained by A_i.

5.2 Proposed heuristics

To enhance the efficiency of both approaches, DRAC and GDRAC, we integrate in their corresponding protocols the following new heuristics based on the below two properties and this in order to decrease the number of constraint checks without loss of correctness. Before describing the two properties, we will define first the new notion of *temporally arc inconsistent*. Note that the condition of the following first property was used AC-7 [8] for binary constraints as an additional condition to perform a constraint check.

Definition 36 *For any binary CN, a subset of values of a variable X_i, $D'(X_i) \subseteq D(X_i)$ is **temporally arc inconsistent** for a value v_{j_k} of a variable X_j if and only if the first support of each value $v_{i_l} \in D'(X_i)$ in $D(X_j)$, $v_{j_m} \in D(X_j)$, $v_{j_k} \prec_{lo} v_{j_m}$.*

Property 1 *For each binary constraint C_{ij}, for each candidate value $v_{j_k} \in D(X_j)$, "hide" from the domain of the related variable $D(X_i)$ all the values v_{i_l} such that v_{i_l} is temporally arc inconsistent for v_{j_k} , and vice versa.*

Proof.
We will simply show that each hidden value v_{i_l} is not compatible with the value v_{j_k}. Therefore, we suppose that $\exists$ *t'* $\in C_{ij}$ such that *t'*[index(C_{ij}, X_j)]= v_{j_k} and *t'*[index(C_{ij}, X_i)]= v_{i_l}. If this tuple exists then *t'* $\prec_{lo}$ *t* (*t* $\in C_{ij}$ such that *t*[index(C_{ij}, X_j)]= v_{j_m} and *t*[index(C_{ij}, X_i)]= v_{i_l} because $v_{j_m} > v_{j_k}$. So *t* cannot be the first tuple support of v_{i_l}.

■

To illustrate the principle of the proposed properties, consider the simple example in Figure5.1 formed by two variables X_1 and X_2 related by a binary constraint C_{12}. An arrow from a value v_{1_k} for X_1 to a value v_{2_l} for X_2 indicates that a check of the consistency between these two values has been computed while seeking for the first support. The bidirectionality property is used to infer the support of the other values. Assume that the value 3 of X_1 is no more viable (due to another constraint). Then we should seek for another support for X_2=2 in X_1. This requires 5 constraint checks. However, we can save 4 constraint checks by using property 1. (for binary constraint). The idea consists on hiding from the domain of X_1 all the values that have as first support v_{2_l} such that $v_{2_l} < 2$, while searching a new support for the value 2 of X_2.

The following definition generalizes the notion of temporally arc inconsistent to any general CN.

Definition 37 *For any general CN, for each n-ary constraint $C_{ij...}$, for each variable $X_h \in Var(C_{ij...})$, a subset of values $D'(X_h) \in D(X_h)$ is generalized temporally arc inconsistent for the value $v_{k_l} \in D(X_k)$ with $X_k \in Var(C_{ij...})$ if and only if each value $v_{h_f} \in D'(X_h)$ satisfies the two following conditions:*

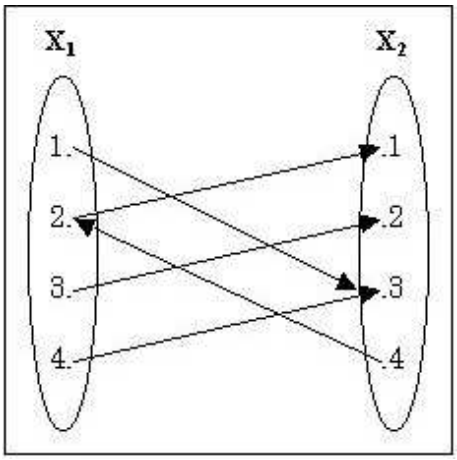


Figure 5.1. Example of binary constraint. Each arrow illustrates the directions of the constraint checks performed in order to seek for the first support for each variable/value.

1. *the first tuple support of* v_{h_f}, $t \in C_{ij...}$, *t[index(*$C_{ij...}$, X_h*)] =* v_{h_f} *and t[index(*$C_{ij...}$, X_k*)] =* v_{h_m} *and* $v_{h_f} \prec_{lo} v_{h_m}$,

2. $\forall\, v_{h_f} \in$ t*; s* $\in$ *1..* $|X(C_{ij...})|$ *such that s* $\neq$ h *and* s $<$ k, v_{h_f} *= last(D(*X_i*)).*

Property 2 *For each n-ary constraint* $C_{ij...}$*, for each candidate value* $v_{k_l} \in D(X_k)$ *with* $X_k \in Var(C_{ij...})$*, "hide" from the domains of all the related variables* X_h, $X_h \in Var(C_{ij...})$ *and* h $\neq$ k*, all the values* v_{h_f} *such that* v_{h_f} *is generalized temporally arc inconsistent for* v_{k_l}.

Proof.
For the first part of this Property2. the proof is similarly to the first Property1. However, the second condition is added in order to guarantee that *t* is the highest tuple in $C_{ij...}$ and no tuple *t'*, that contains *t'*[index($C_{ij...}$, X_k)] = v_{k_l}, *t'*[index($C_{ij...}$, X_h)] = v_{k_l} and $t \prec_{lo} t'$, exists.

■

5.3 Global constraint-agents interactions

The main objective of both approaches is to transform a CSP *P* (X, D, C) into another CSP *P'*(X, D', C) equivalent. *P'* is obtained as a result of interactions between the Constraint agents which are trying to reduce their domains D' $\subseteq$ D.

Before detailing these interactions and the underlying global dynamic, we present the communication protocol, the data structures and the basic primitives relative to an agent A_i.

5.3.1 Communication protocol

The communication protocol is based on the two following message passing primitives.

- *SendMsg*(*Sender*, *Receiver*, *"Message"*) where *Receiver* can be more than one receiver.

- *GetMsg*() extracts the first message from the *mailBox*.

As regards to the exchanged messages, the Multi-Agent dynamic involves three types of messages (without considering those relative to the detection of the equilibrium state) namely:

- *"Start"* message, sent by the interface to all the agent, Γ, in order to activate them,

- *"ReduceDomains"* message, sent by a Constraint agent A_i to its acquaintances Γ^{A_i} in order to propagate its deleted values.

- *"StopBehavior"* message sent by a Constraint agent A_i, which has a domain wipe-out, to the interface.

- *"StopLocalBehavior"* message sent by the interface to all the agents, Γ, of the system to make them stop their local behavior.

We must note that, as we have mentioned above, all the messages are received in the same order in which they were sent, except for those used to detect the termination of the system. Therefore, we concede higher priorities to these latest messages. Thus they can overtake any message in the queue (*mailbox*). This feature leads to a quicker termination of the whole process.

5.3.2 Common data structures and basic primitives

In more detail, the common required data structure and basic primitives for both approaches, are the following:

- *AcqConst*^{A_i}[X_k]=$\{A_j \in \Gamma^{A_i} / X_k \in Var(C_{ij...}^{A_i}) \cap Var(C_{ij...}^{A_j})\}$, is the ordered set of all the Constraint agents sharing the variable X_k with the agent A_i.

- D^{A_i}=$\{D^{A_i}(X_k) / X_k \in Var(C_{ij...}^{A_i})\}$ represents the local view of the domains $D^{A_i}(X_k)$ of all the variables maintained by A_i. Each domain is supposed to be totally ordered. For all $X_k \in Var(C_{ij...}^{A_i})$, $D^{A_i}(X_k)$ is called the occurrence of $D(X_k)$. Note that some occurrences of a given $D(X_k)$ may be different, but all occurrences of $D(X_k h)$, for all $k \in \{1, \ldots, n\}$, must be identical when the full global arc-consistency is reached. At this step, let us refer to the final obtained domain $D^{A_i}(X_k)$ by $f D^{A_i}(X_k)$.

- *TupleSupport*^{A_i} is the set of tuple t where t is a vector of consistent assignments of the variables involved in $C_{ij...}^{A_i}$ with t = (v_{i_k}, v_{j_l}, ..., y), r is the arity of $C_{ij...}^{A_i}$, and (v_{i_k}, v_{j_l},

. . .) satisfies $C_{ij...}$. The parameter $y \in \{0, 1, \ldots, r\text{-}1\}$ is used to indicate that (v_{i_k}, v_{j_l}, . . .) is the "first" tuple support for the value *t*[*y*+1].

Let's recall that tuples are ordered according to the natural lexicographic order $\prec_{lo}$ such that for each $\{t, t'\} \in C_{ij...}^{A_i}$, $t \prec_{lo} t'$ if and only if there exist *k* such that *t*[1..*k*-1]=*t'*[1..*k*-1] and *t*[*k*]< *t'*[*k*].

- *IncValue*^{A_i}[X_k]=$\{v_{k_m} \in D^{A_i}(X_k)$ / there is not any tuple $t \in C_{ij...}^{A_i}$ such that *t*[*index*($C_{ij...}^{A_i}$, X_k)] = v_{k_m} and *t* is valid$\}$, represents the set of all current inconsistent values for X_l belonging to $Var(C_{ij...}^{A_i})$.

- *HD*^{A_i}[X_k]=$\{v_{k_l}$ / $v_{k_l} \in D^{A_i}(X_k)$, $X_k \in Var(C_{ij...}^{A_i})$, and v_{k_m} verifies Property2$\}$, represents the new current domains after hiding some temporary arc inconsistent values.

- *ReviseValue*^{A_i} is the set of all current values that should be revised.

The common basic primitives are:

- *addTo*(l, e): add e to the set l; e is added to the end of l.

- *first*(l): returns the first element in the set l if $|l| > 1$; otherwise returns *nil*;

- *last*(l): returns the last element in the set l if $l \neq \emptyset$; otherwise returns *nil*;

- *next*(e, l): returns the first element e' occurring after e in the set l if $e' \neq$*Last*(l); otherwise returns *nil*;

- *delete*($l1$, $l2$): for each element $e \in l2$, if $l1$ contains e then e should be removed from $l1$; otherwise nothing to do.

- *hideFrom*(l, e, var): remove *temporally* all the elements e' from l according to some conditions related to e ;

- *SearchNewSupport*(e, *t*): returns the *smallest*[2] tuple support *t'* $\in C_{ij...}^{A_i}$ such that *t*$\prec_{lo}$ *t'* and *t'*[$index(C_{ij...}^{A_i}$, X_j)]=e. If *t*=*nil*, then this primitive will return the *first* tuple support.

5.3.3 Agent-based protocol

The DRAC and G-DRAC approaches exhibit almost the same global dynamic. The main dissimilarity rests in the fact that DRAC is only for binary constraints, while GDRAC is a generic distributed approach for any general CN.

At the initial state, the Interface agent creates all the Constraint agents, Γ, and activates them. Each agent A_i maintaining a constraint $C_{ij...}^{A_i}$ reduces the domains (D^{A_i}) of its own

[2]There is no other tuple support *t"* such that *t* $\prec_{lo}$ *t"* and *t"* $\prec_{lo}$ *t'* and *t"*[index($C_{ij...}^{A_i}$, X_j)]= v_{j_k}

variables, i.e. $\forall\ k \in \{1, \ldots, r\}$; r=$|Var(C_{ij\ldots}^{A_i})|$ and $X_k \in Var(C_{ij\ldots}^{A_i})$ by computing local viable values for each variable. For achieving this, $C_{ij\ldots}^{A_i}$ looks for one tuple support (the first one) for each value v_{k_m} of each of its variables X_k. When the first support *t*, that satisfies $C_{ij\ldots}^{A_i}$ and *t*[index($C_{ij\ldots}^{A_i}$, X_k)] $\in D^{A_i}(X_k)$, is found, then (v_{i_k}, v_{j_l}, ..., (*k*-1)) is added to the list of tuple supports *TupleSupport*^{A_i}.

We must note that v_{i_k}, v_{j_l}, ... are the first values support for v_{k_m} but they are also values support for each other by applying the multidirectionality[3] property of constraint relations. A value v_{k_m} is deleted from $D^{A_i}(X_k)$ if and only if it has no viable tuple support. Each obtained set of deleted values for a variable should be announced *immediately* to the concerned acquaintances in order to save fruitless consistency checks for these values.

Each agent that received this message starts processing it by updating the domains of its variables while deleting non-viable received values. At the end of this computation, it updates computed support information by deleting all non-viable tuples. In the case in which v_{k_m} is an inconsistent value, the agent determines first all the tuple *t* such that *t*[index($C_{ij\ldots}^{A_i}$, X_k)]= v_{k_m}. Then checks the existence of another viable tuple support *t'* in *TupleSupport*^{C_j} for each value $a_{j_l} \in t$ ($a_{j_l} \in D^{A_i}(X_j)$ and $k \neq j$). If such tuple t' does not exist, the agent searches for a new tuple support. Therefore the agent starts first by "hiding" all the values that are incompatible with v_{k_m} from the domains of all the related variables using the aforementioned Property2 (*resp*. Property1 for binary constraints). This allows us to reduce the number of constraint checks.

Second, the agent looks for another tuple support for each value v_{j_l} according to *y* and using the new domains. If *y*=(*h*-1) then the search must be done from the smallest tuple *t'* (according to the predefined order) such that $t \prec_{lo} t'$ (as AC-6). Otherwise, it looks for a support from the scratch i.e., the first (smallest) tuple in C_j. This can lead to a new values deletion and by consequence, to new out going messages. So reducing domains on an agent may, consequently, cause an eventual domain reductions on another agent.

Hence, the same process must resume until all the arc-consistent values have been deleted from the domains of all the variables. This state is known as a stable equilibrium state, in which all the agents of the system are satisfied. An agent is satisfied when it has no more reductions to do on its variables' domains or when one of its reduced domains wipes out. However, it is clear that the satisfaction state of a single agent is not a definitive state. Indeed, if there exists at least one unsatisfied agent, it may cause the unsatisfaction of other Constraint agents; this is due to the propagation of constraints.

An agent is satisfied when it has no more reduction to do on its variable domains or when one of its reduced domain wipes-out. However, it is clear that this satisfaction state is not definitive. Indeed, if there exists at least one unsatisfied Agent, it may cause the unsatisfaction of other Constraint agents and this is due to the propagation of constraints.

We should emphasizes that this dynamic allows a premature detection of the failure i.e.

[3]or applying the bidirectionality property in case of binary constraints.

absence of solutions, and this when the domain of at least one variable wipe-out. Hence, with regard to agents' behavior, each Constraint should continue its local behavior until attaining its satisfaction state[4]:

- When one of its domains wipes out. In this case, it asks the interface to stop the whole process and to communicate the failure result to the user.
- When all possible local reductions have been accomplished, while taking into account the just-received messages containing the values deleted by the other Constraint acquaintances. In this case, it updates its internal state.

Otherwise, i.e., in the case of unsatisfaction behavior, it sends a message containing inconsistent values to the concerned acquaintances.

The Interface agent is satisfied when all the agents are satisfied or when it has received a failure message; then it makes all the agents stop their local behavior, and communicates the obtained result to the user. Otherwise, i.e., in the case of unsatisfaction behavior, it checks the system state using the algorithm described by Lamport and Chandy [19].

5.4 Theoretical analysis

5.4.1 Correctness

The correctness of the proposed hybrid method (the two approaches) relies, in great part, on the correctness of the DRAC protocol. The objective of this subsection is to exhibit the accuracy of the proposed DRAC approach[5] and to show that it leads to full global arc-consistency. For this result, we must prove the following assertions:

- For all $X_h \in X$, $h = \{1, \ldots, n\}$, for all $\{A_i, A_j\} \in \Gamma$ such that $X_h \in Var(C_{ij\ldots}^{A_i}) \cap Var(C_{ij\ldots}^{A_j})$, $i \neq j$, $fD^{A_i}(X_h) = fD^{A_j}(X_h)$.
- For all $X_h \in X$, $h \in \{1, \ldots, n\}$, for all $A_i \in \Gamma$, for all $X_h \in Var(C_{ij\ldots}^{A_i})$, for all $v_{k_l} \in fD^{A_i}(X_k)$, v_{k_l} is arc-consistent.
- For all $X_h \in X$, $h \in \{1, \ldots, n\}$, for all $A_i \in \Gamma$, for all $X_h \in Var(C_{ij\ldots}^{A_i})$, for all $v_{h_l} \in D^{A_i}(X_h)$, if v_{h_l} is viable then $v_{h_l} \in fD^{A_i}(X_h)$.

In fact, the first assertion concerns the process of deleted values propagation. Since for all $A_i, A_j \in \Gamma$, $i \neq j$ such that $X_h \in Var(C_{ij\ldots}^{A_i}) \cap Var(C_{ij\ldots}^{A_j})$ and consequently A_j belongs to the acquaintances of A_i, $A_j \in \Gamma^{A_i}$ (and conversely, $A_i \in \Gamma^{A_j}$), and since all the messages are received in a finite period of time and in the same order as they were sent, A_i (*resp.* A_j)

[4]This state of satisfaction is based on local knowledge to detect whether the local goal is achieved or not. However, this state may be temporal if the global goal is not accomplished.

[5]As for the G-DRAC, its protocol is based essentially on DRAC protocol.

must be informed by each deleted value[6]. Then the agents will have the same final domains $f\mathrm{D}^{A_i}(\mathrm{X}_h)$ and $f\mathrm{D}^{A_j}(\mathrm{X}_h)$.

The second assertion concerns the correctness of the *ReduceDomains:for:* procedure. Each time, the deletion of a value (from $\mathrm{D}^{A_i}(\mathrm{X}_h)$) leads to a non-viable value in the domain of a variable X_h. The agent A_i sends a message to all the concerned acquaintances $\mathrm{A}_j \in \Gamma^{A_i}$, asking them to update their X_h domain. So, all the non-viable values are deleted from the domains of all the agents. Thus, each value remaining in the final domain of each variable is arc-consistent.

For the third assertion, there are two cases in which a value v_{h_l} is deleted from the domain of a variable X_h. The first is that the agent A_i has detected that v_{h_l} has no support in *at least* one variable X_k ($k \neq h$) and $\mathrm{X}_k \in$ *Var*($\mathrm{C}^{A_i}_{ij\ldots}$). Therefore, v_{h_l} is a non-viable value and must be discarded. The second case is when the agent A_i has received a message to update the domain of X_h by deleting the value v_{h_l}. Thus, this value has been detected as non-viable by the agent which sent the message. Consequently, only the non-viable value will be deleted.

5.4.2 Termination detection

The dynamic of DRAC approach stops when the system reaches its stable equilibrium state. At this state, all the agents are satisfied. An agent is satisfied when it has no more reductions to do on its variable domains or when one of its related new reduced domains is wipe-out. In the first case, the global goal of all the agents is reached, i.e., achieving global full arc consistency.

However, in the second case, the problem is inconsistent i.e. no instantiation satisfies all the constraints. The detection of the stable equilibrium state in a distributed system can be achieved by taking a snapshot of the system, using the well known algorithm of [19], a state where all agents are waiting for a message and there is no message in the transmission channels. If all the agents of the system are in the state of waiting, and there exists only one agent A_i which has deleted one value v_{h_l} from the domain of one of its variables (X_h $\in$ *Var*($\mathrm{C}^{A_i}_{ij\ldots}$). We assume that this agent shared this altered variable with another agent A_j. The latter must be informed of the loss of the value v_{h_l} in order to propagate the constraints. Hence, there is a message in transit for it, which invalidates our transmission hypothesis.

We notice that, the cost of the termination process can be mitigated by combining snapshots messages with our protocol messages.

[6]Let us recall that the deleted values must be immediately transmitted to the concerned acquaintances.

5.4.3 Spatial and temporal complexities

Let us consider a CSP *P* having *n* for the total number of variables, *d* for the size of the variable domains and *e* for the total number of constraints. The number of agents is *e*. If we consider a fully connected constraint network, we will have *e-1* acquaintances for each Constraint agent. Each agent A_i maintains a list $TupleSupport^{A_i}$ of supports, with the size of *2d-1* in the worst case (for only binary constraints). Since there are *e* agents, the total amount of space is (*2d-1*)*e* (for a fully connected graph, *e* will be set to *n(n-1)/2*, in the worst case). So the space needed for DRAC is (*n(n-1)/2*)*(*2d-1*) $\simeq$O(n^2d). This space is the same as that of AC-7 one's.

The worst case in the execution time of a distributed algorithm occurs when it proceeds with a sequential behavior. For our protocol, this occurs when only one value is deleted at a time, leading to *nd* successive deletions. Our approach is composed of two steps. The first is the initializing step, in which each agent performs d^2 operations to generate the support sets.

In step 2, for each deleted value, the agent will perform O(d^2) operations to search another support for this value. Thus, each agent performs O(d^2) operations. So the total time complexity of DRAC (with *e* agents and *nd* successive deletions), in the worst case, is O(end^3). This complexity is equal to that of DisAC-9 down to the number of variables.

Regarding GDRAC complexity, each agent A_j in the system also maintains a list of tuples support $TupleSupport^{A_j}$. Each value should have at most one tuple support, then the total number of tuples is at most *rd* with *r* for the maximal arity for each constraint. Each tuple has (*r*+1) values. Since there are *e* agents, the total amount of space for each agent is (*rd*)(*r*+1). So the space needed for each agent, in the worst case, is O(r^2d), the same as GAC-Schema one's. Each agent performs d^r operations to generate the tuples support sets, and for each deleted value the agent will perform O(d^{r-1}) operations to search for another tuple support for this value. While the total time complexity of GDRAC, in the worst case, is O(end^r).

5.5 Experimental comparative evaluation

The implementation of the proposed hybrid method was developed with Actalk, an object based on concurrent programming language in the Smalltalk-80 environment. In this language framework, an agent is implemented as an actor having the Smalltalk object structure enriched by an ability to send/receive messages to/from its acquaintances, buffering the received messages in its own *mailbox*. The experimentations were performed over randomly generated instances and using five parameters: *n* is the number of variables, *d* is the domain size of each variable, *r* is the maximal arity of the constraints, *p* is the graph connectivity (the proportion of constraint in the network, *p*=1 corresponds to the complete graph) and *q* is the constraint looseness (the proportion of allowed pairs of values in a constraint). We have performed two groups of experiment to evaluate the two proposed approaches of our hybrid method.

In the first group, the efficiency of the DRAC approach is assessed through a comparison with AC-7. Let us recall that AC-7 is the best centralized algorithm to enforce arc-consistency on any binary CN. This algorithm is used as a witness algorithm to evaluate the performance and efficiency of DRAC.

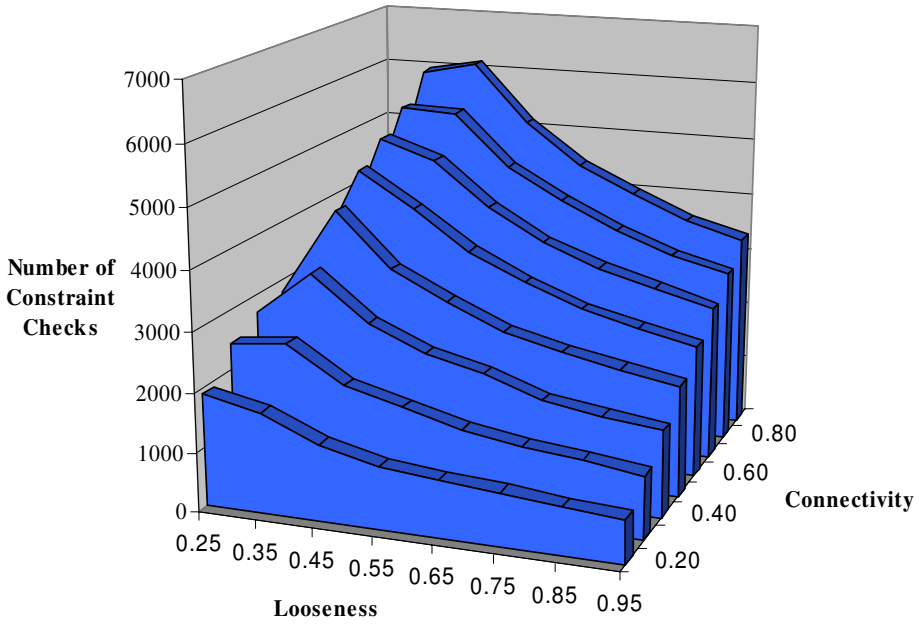


Figure 5.2. DRAC results in mean number of Constraint Checks for constraints in intension on Pentium III (35 instances are generated for each set of $\langle p; q \rangle$ parameters).

Each used sample is designed on the base of the following parameters: n = 20, d = 10, p=$\{0.2, \ldots, 0.9\}$ per step of 0.1 and q=$\{0.25, \ldots, 0.95\}$ per step of 0.1 also. For each pair $\langle p, q \rangle$, 35 random examples were generated. Figures 5.6 and 5.7 show that AC-7 and DRAC require almost the same number of constraint checks especially for over-constrained problems. While for CPU time, Figures 5.4 and 5.5 show that AC-7 requires more time than DRAC especially for large problems.

In the following we will try to focus essentially on the most complex problems for the same above parameters. Therefore, we directed our attention to evaluate the performance of DRAC approach for the most hard problems belonging to the transition phase[7]. Hence, we brought out two versions of DRAC: DRAC-Ext and DRAC-Int for constraints in extension and in intension respectively. Figure5.6 shows that AC-7 performs almost the same number of constraint checks as DRAC-Int, while for CPU time, DRAC-Ext requires a lower time than the two others (Figure5.7). This result shows that DRAC is especially worthwhile for

[7]Problems for which establishing arc-consistency sometimes succeeds sometimes not.

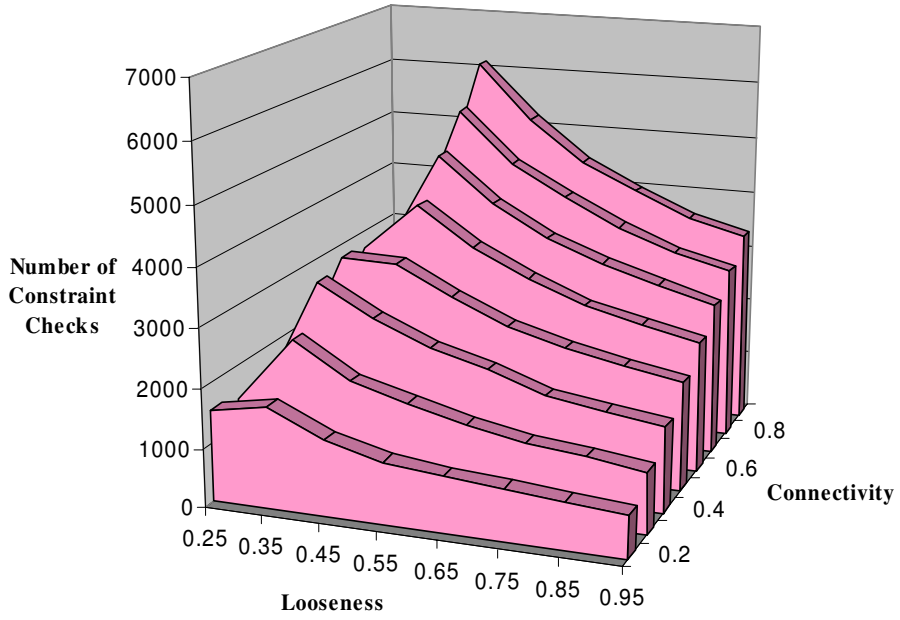


Figure 5.3. AC7 results in mean number of Constraint Checks for constraints in intension on Pentium III (35 instances are generated for each set of $\langle p; q \rangle$ parameters).

constraints in extension, due to the fact that the DRAC first step (initialization) explores the stored relation of each constraint only once in order to generate the required sets of supports (*TupleSupport*^{A_i} sets).

The DRAC protocol allows us to perform constraints according to their type of representation (how they are expressed).

The second group of experiments was designed to test the performance of GDRAC for general CN. We used GAC-7 [11] as a witness approach to appraise the efficiency of the results of GDRAC. Another set of instances, was randomly generated according to the following parameters, n=20; d=10; r=3; $p \in \{0.2, ..., 0.9\}$ with step of 0.1 and $q \in \{0.3, 0.38, 0.41, 0.43, 0.45, 0.46, 0.47, 0.48\}$. For each $\langle p, q \rangle$, 10 CNs instances were also tested.

The results reported below represent the average of the obtained number of constraint checks. We have also implemented four different versions of our approach in order to check its efficiency in handling any type of constraints, i.e., those expressed in extension or in intension.

Figure5.8 represents the result of the two first versions of the proposed approach, GDRAC-Ext1 and GDRAC-Ext2, for constraints given in extension. The main objective of this experiment is to highlight the impact of immediately processing each received message. Thus, in the first one, each agent processes out the received message after completing its current

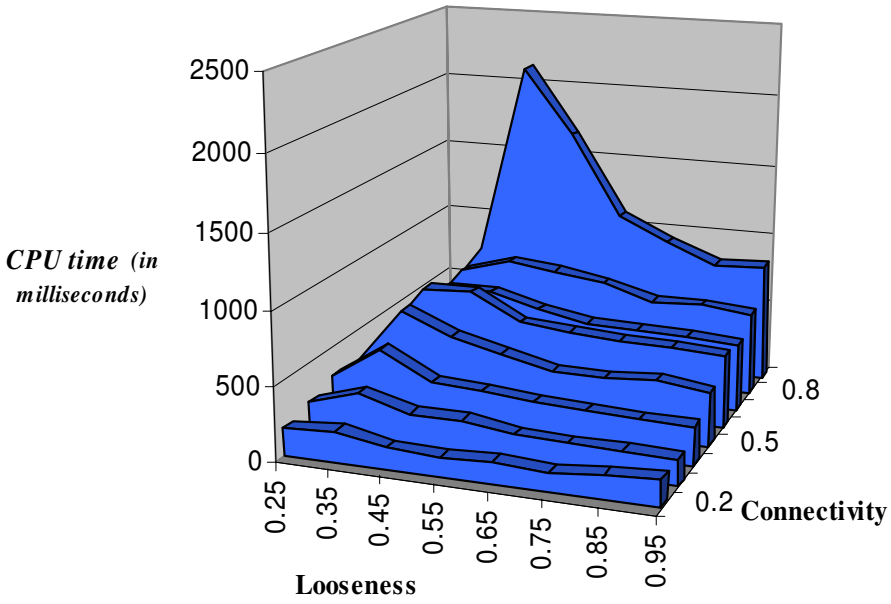


Figure 5.4. DRAC results in mean CPU time for constraints in intension on Pentium III (35 instances are generated for each set of $\langle p; q \rangle$ parameters).

work. In contrast, in GDRAC-Ext2, each agent tries to execute each received message immediately. GDRAC-Ext1 entails more ccks than GDRAC-Ext2, especially for dense instances. This can be justified by the fact that giving the agent relevant information in time allows it to immediately update and propagate the consequence of the received message, thus saving many fruitless efforts, i.e., re-checking the viability of some values already detected as inconsistent by another agents.

We carried out a second experiment on GDRAC to check the efficiency of our approach for the most hard type of constraints (constraint in intension where no particular semantic is known). We propose in this experiment two other versions of GDRAC, GDRAC-Int1 and GDRAC-Int2. The objective is to check the usefulness of the proposed property of temporal inconsistency for n-ary CN (Section5.2). Figure5.9 shows that the number of ccks performed by GDRAC-Int2, i.e., GDRAC-Int2 with property, converge to the number given by the best centralized approach, GAC-7.

The difference in ccks between GDRAC-Int1 and GAC-7 can be vindicated by the fact that for GDRAC-Int1, each constraint is represented by an agent and each variable can be shared by several agents. All the agents will perform local arc-consistency in parallel, which leads to the fact that each value can be detected as inconsistent by several constraints at the same time, leading to more ccks. However, for the centralized approach, if a value is detected inconsistent by one constraint, then it will not be checked by the other constraints (sequential

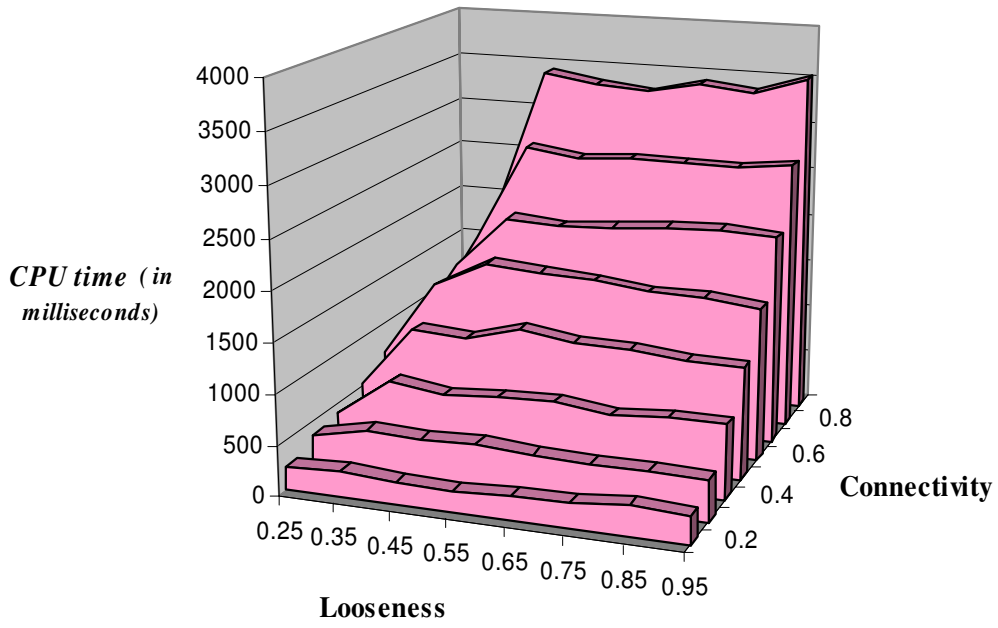


Figure 5.5. AC7 results in mean CPU time for constraints in intension on Pentium III (35 instances are generated for each set of $\langle p; q \rangle$ parameters).

treatment). To this end, we can say that the proposed property allows GDRAC to decrease as maximum the number of constraint checks in order to converge to the required number for GAC-7. We should not forget that for some problems it is not possible to solve the problem by only one agent, for many reasons, such as data privacy and security problems.

We also tested the behavior of our approach toward inconsistent problems. The main objective of this second type of experimentation is to evaluate the percentage of deleted values required for detecting the insolubility of a CN. Table 1 shows the ratio of the percentage of the obtained results of GDRAC divided by those of GAC-7, in the mean of the percentage of deleted values and CPU time (in milliseconds). In most cases, GDRAC prunes a few more values than GAC-7 to prove insolubility; in the others, it is almost the same.

At first glance, this result does not seem good, especially if this pruning process needs more time to be accomplished. However, the CPU time needed for our approach is less than that of GAC-7 (ratio$<$1). In the majority of cases GDRAC requires about half of the time needed by GAC-7. In addition, the difference in the percentage of the deleted values can be justified by the fact that for our approach all the constraints should be activated at the same time, leading to more deletions, even if the problem is inconsistent. As for GAC-7 the insolubility of the problem can be detected at the beginning without invoking all the constraints.

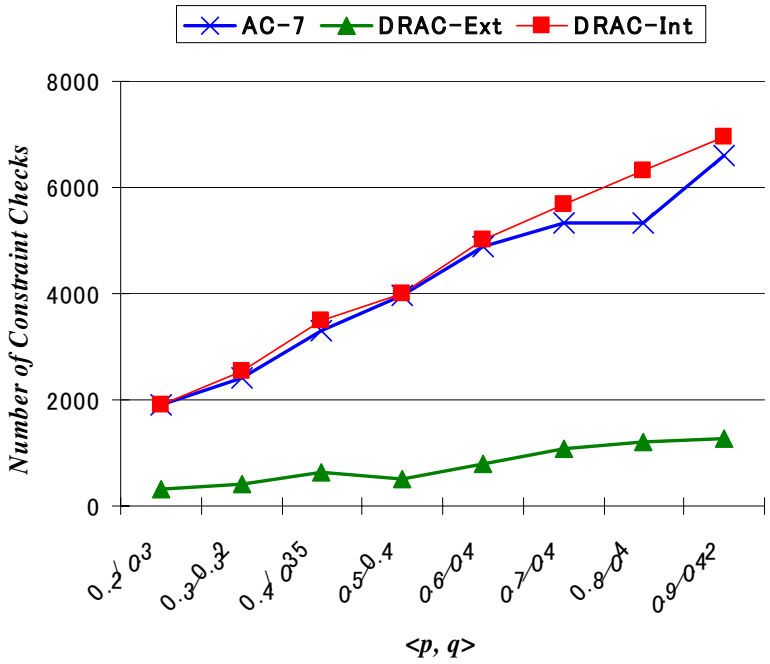


Figure 5.6. DRAC-Int and DRAC-Ext vs. AC7 Results in mean number of Constraint Checks on Pentium III (10 instances are generated for each set of $\langle p; q \rangle$ parameters).

We should note that for our approach, even though all the constraints are called upon, the local decisions are quickly propagated globally to prove the inconsistency, especially for constraints in extension.

5.6 Summary

In this chapter we suggested a new agent-based hybrid method incorporating two proposed approaches. The common model, used by these approaches, consists of a set of Constraint agents, each having a local goal, in communication by exchanging asynchronous point-to-point messages. These messages contain the local inconsistent values in order to help the agents to reduce the domains of the variables that they involve. This process is performed until a stable equilibrium state is reached and corresponds to a failure relative to an absence of solutions or to the achievement of the global goal. Thus, this state is obtained as a side effect of the interactions among the Constraint agents, whose behaviors are simple and reactive.

As we associate an agent per Constraint, the dual constraint-graph is proved to be appropriate for representing any general CN. Consequently, any generalized CSP can be naturally and directly handled (without any non-binary $\Longrightarrow$ binary transformation).

In this work, we proposed new properties to improve the efficiency of these approaches. The experimental comparative evaluation, of the two approaches, shows the efficiency of the

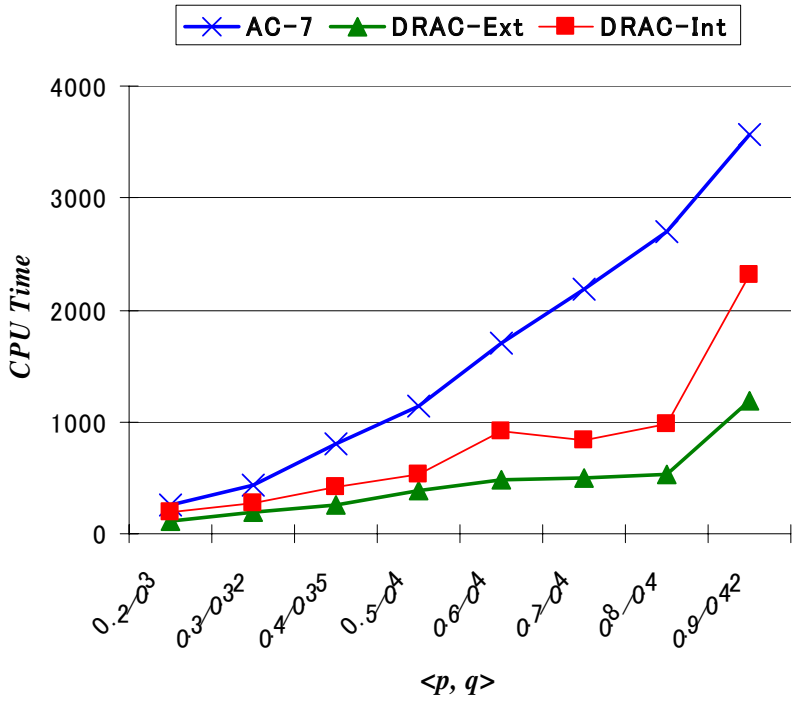


Figure 5.7. DRAC-Int and DRAC-Ext vs. AC7 Results in mean number of CPU time on Pentium III (10 instances are generated for each set of $\langle p, q \rangle$ parameters).

DRAC approach especially for constraints in extension, the high performance of GDRAC as a distributed arc-consistency technique for any general CN. These two approaches have been published in [4, 5, 6, 10] and under reviewing in the *International Journal of Artificial Intelligence Tools* [1].

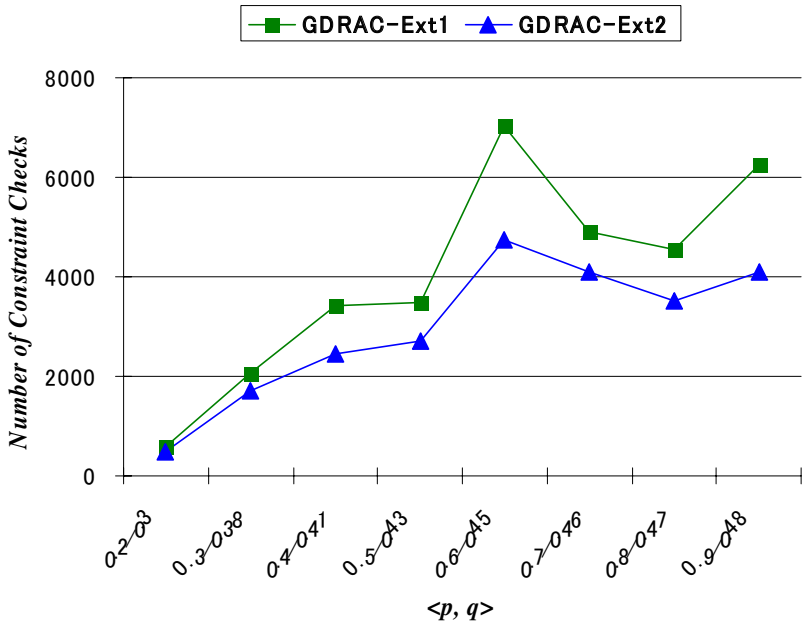


Figure 5.8. G-DRAC-Ext1 vs. G-DRAC-Ext2 results in mean number of Constraint Checks for constraints expressed in extension.

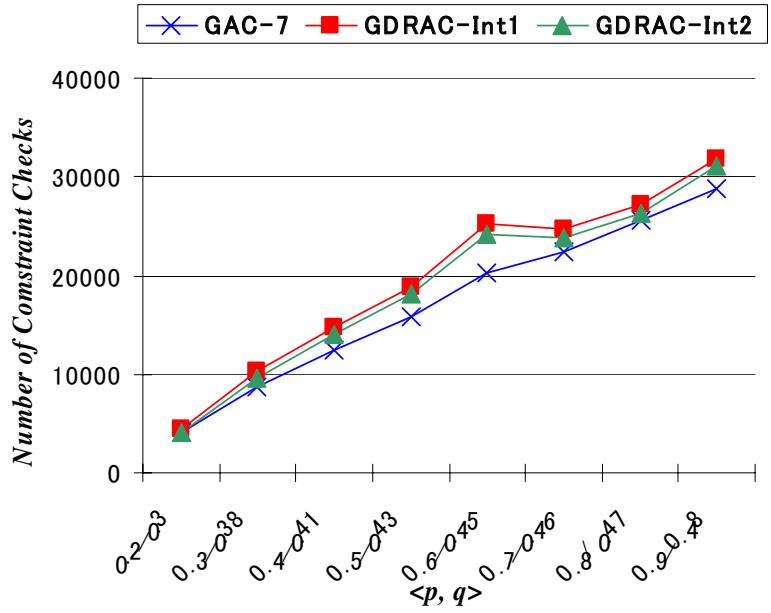


Figure 5.9. G-DRAC vs. GAC7 results in mean number of Constraint Checks for constraints expressed in intention.

Algorithm 1 Start message executed by each Constraint agent A_i

begin

1: **for all** $X_k \in Var(C_{ij...}^{A_i})$ **do**
2: *IncValue*$^{A_i}[X_k] \leftarrow D^{A_i}(X_k)$;
3: **end for**
4: /* *We assume that all the values are initially non-viable* */
5: Choose randomly one variable to process $X_k \in Var(C_{ij...}^{A_i})$;
6: **for all** ($v_{k_l} \in D^{A_i}(X_k)$ **such that** $v_{k_l} \in$ *IncValue*$^{A_i}[X_k]$) **do**
7: *t'* $\leftarrow$ *SearchFirstSupport*(v_{k_l}, *nil*);
8: **if** $t \neq$ *nil* **then**
9: addTo(*t*, *index*($C_{ij...}^{A_i}$, X_k)-1);
10: addTo(*TupleSupport*^{A_i}, *t*);
11: delete(*IncValue*$^{A_i}[X_k]$, v_{k_l});
12: **end if**
13: **end for**
14: **for all** $X_k \in Var(C_{ij...}^{A_i})$ **such that** *IncValue*$^{A_i}[X_k] \neq \emptyset$ **do**
15: delete($D^{A_i}(X_k)$, *IncValue*$^{A_i}[X_k]$);
16: **if** ($D^{A_i}(X_k)=\emptyset$) **then**
17: *sendMsg*(*Self*, Interface, "*StopBehavior*");
18: **for all** $X_k \in Var(C_{ij...}^{A_i})$ **such that** *IncValue*$^{A_i}[X_k] \neq \emptyset$ **do**
19: **for all** $A_j \in$ *AcqConst*$^{A_i}[X_k]$ **do**
20: *sendMsg*(A_j, Self, "*ReduceDomains*:*IncValue*$^{A_i}[X_k]$ *for*:X_k");
21: **end for**
22: **end for**
23: **end if**
24: **end for**
25: Process next variable;
26: /* *Return to step 3. to choose another variable* */

Table 5.1. Results obtained in ratio of the percentage of deleted values and CPU time

	$\langle$**0.2; 0.2**$\rangle$	$\langle$**0.3; 0.3**$\rangle$	$\langle$**0.4; 0.35**$\rangle$	$\langle$**0.5; 0.4**$\rangle$
% **Del Values**	1.05	0.86	1.03	1.15
CPU Time	0.46	0.61	0.46	0.52
	$\langle$**0.6; 0.4**$\rangle$	$\langle$**0.7; 0.42**$\rangle$	$\langle$**0.8; 0.43**$\rangle$	$\langle$**0.9; 0.44**$\rangle$
% **Del Values**	1.05	1.05	0.98	0.99
CPU Time	0.52	0.51	0.61	0.92

Algorithm 2 Propagation procedure executed by each Constraint agent A_i

ReduceDomain:*delVal* ***for***:X_k

1: **for all** ($v_{k_m} \in$ *delVal*) **such that** ($v_{k_m} \in D^{A_i}(X_k)$) **do**
2: delete($D^{A_i}(X_k)$, v_{k_m});
3: **end for**
4: **for all** $t \in$ *TupleSupport*^{A_i} **such that** t[*index*($C^{A_i}_{ij...}$, X_k)]=v_{k_m} **do**
5: delete(*TupleSupport*^{A_i}, t);
6: **for all** $v_{k_h} \in t$ **such that** $h \in \{1, \ldots, |Var(C^{A_i}_{ij...})|\}$ and $h \neq m$ **do**
7: **if** (last(t)= (*index*($C^{A_i}_{ij...}$, X_k)-1)) **then**
8: addTo(*ReviseValue*^{A_i}[*index*($C^{A_i}_{ij...}$, X_k)], (v_{k_h}, t));
9: **else**
10: addTo(*ReviseValue*^{A_i}[*index*($C^{A_i}_{ij...}$, X_k), (v_{k_h}, *nil*)]);
11: **end if**
12: **end for**
13: **end for**
14: **for all** (v_{k_h}, t) $\in$ *ReviseValue*^{A_i} **do**
15: **if** ((*Check*:v_{k_h} *for*:X_k) = false) **then**
16: $HD^{A_i} \leftarrow$ *hideFrom*:D^{A_i} *for*:v_{k_h} *of*:X_k;
17: *t'* $\leftarrow$ *SearchNewSupport*:v_{k_h} *from*:*t in*:HD^{A_i};
18: **if** (t' = $\emptyset$) **then**
19: delete($D^{A_i}[X_k]$, v_{k_h});
20: **if** ($D^{A_i}[X_k]$=$\emptyset$) **then**
21: *SendMsg*(*Self*, Interface, "*StopBehavior*");
22: addTo(*IncValue*^{A_i}[X_k], v_{k_h});
23: **else**
24: addTo(t', index($C^{A_i}_{ij...}$, X_k)-1);
25: addTo(*TupleSupport*^{A_i}, t);
26: **end if**
27: **end if**
28: **end if**
29: **end for**
30: **for all** ($X_j \in Var(C^{A_i}_{ij...})$ **such that** *IncValue*^{A_i}[X_j]$\neq \emptyset$) **do**
31: **for all** ($A_k \in$ *AcqConst*^{A_i}[X_j]) **do**
32: *SendMsg*(A_k, *Self*, "*ReduceDomain*:*IncValue*^{A_i}[X_j] *for*:X_j");
33: **end for**
34: **end for**

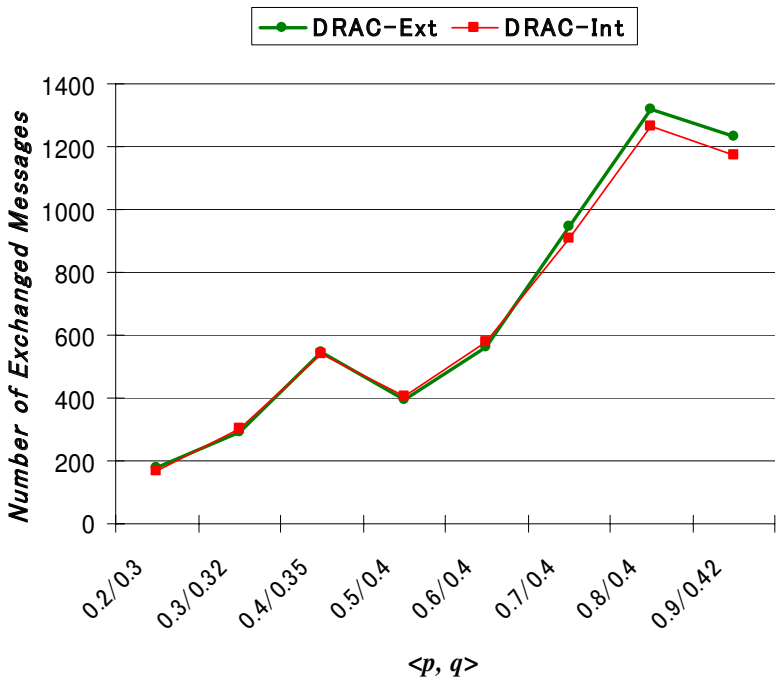


Figure 5.10. GDRAC-Ext1 vs. GDRAC-Ext2 results in mean number of exchanged messages for constraints expressed in intention.

Chapter 6

DRAC^{++} to Enforce more than AC

As mentioned before enforcing AC on some hard CN may be fruitless. The main reason is that the problem might be initially AC. Performing more than AC may prune more values and consequently may enhance better the solving process.

In this chapter we discuss a new approach for performing distributed restricted path consistency property on a binary CN.

6.1 Distributed enforcement of restricted path consistency

In the following, we propose a new property based on RPC that we will use in the proposed protocol in order to prune more inconsistent values from the CN. The main objective is to improve the efficiency of DRAC approach without loss of correctness.

6.1.1 Knowledge inference heuristic

Property 3 *For each path of three variables P_3=$\{X_i, X_j, X_k\}$ of an arc-consistent CN. For each $X_i \in P_3$, for each value $v_{i_l} \in D(X_i)$ and its arc-consistent support*[1] *$v_{j_h} \in D(X_j)$, v_{i_l} is an inconsistent value and consequently should be removed from $D(X_i)$ if and only if:*

- *There is no common support $v_{k_m} \in D(X_k)$ such that TupleSupport$^{A_i}[v_{i_l}]$ = TupleSupport$^{A_j}[v_{j_h}]$ = v_{k_m} with $\{X_i, X_k\}$ maintained by A_i and $\{X_j, X_k\}$ maintained by A_j*[2].
- *For all $v_{j_f} \in D(X_j)$ with $v_{j_f} \neq v_{j_h}$ and for all $v_{k_n} \in D(X_k)$, TupleSupport$^{A_i}[v_{j_f}]$=v_{i_g} with v_{i_g} first support of v_{j_f} and $v_{i_l} \prec_{lo} v_{i_g}$ and TupleSupport$^{A_k}[v_{k_n}]$=v_{i_t} with v_{i_t} first support of v_{k_n} and $v_{i_l} \prec_{lo} v_{i_t}$.*

  Note that by using bidirectionality between two variables X_i and X_j while enforcing AC, we can have knowledge about the first *support of X_i in X_j and not the inverse.*

[1] This support can be the first support or one support, by using bidirectionality, depending on the used order between variables.

[2] *TupleSupport*^{A_i} represents the collected knowledge, concerning arc-consistent pair of values for the variables maintained by A_i, result of performing AC.

Therefore, in case it is not possible to check this condition, we need to perform more constraint checks to verify the inconsistency of a *by applying RPC property as mentioned in the following condition,*

- *The value $v_{j_h} \in D(X_j)$ is the only support of $v_{i_l} \in D(X_i)$ and there is no common value support $v_{k_m} \in D(X_k)$ such that the pair (v_{i_l}, v_{k_m}) and the pair (v_{j_h}, v_{k_m}) are simultaneously arc-consistent.*

The two first conditions of the above mentioned property are used to infer the oneness of supports for the value v_{i_l} to detect whether it is path inconsistent or not without performing extra constraint checks.

6.1.2 DRAC++ multi-agent model

The proposed model for the new approach DRAC^{++} involves (as for DRAC model) two kinds of agents, Constraint agents, Γ, and the Interface agent, communicating by exchanging asynchronous point-to-point messages. For transmission of messages, we assume that they are received in the same order they were sent and in a finite delivering time.

The main goal of DRAC^{++} is to transform any CSP *P* (X, D, C) into another CSP *P'*(X, D', C) equivalent via interactions among the Constraint agents, which are trying to reduce their domains. The underlying new proposed protocol is divided into two steps

- First, enforce arc consistency on the problem (the same as DRAC protocol),
- Second, use the knowledge collected from the previous step to remove some additional values that cannot belong to any solution by enforcing RPC property.

6.1.3 Basic of the enforcing process

At the initial state, the Interface agent creates all the Constraint agents Γ and activates them. Each agent A_i reduces the domains of its own variables by computing local first viable value for each variable.

Let's recall that for each variable X_i, for each value $v_{i_l} \in D(X_i)$, if its *first* support $v_{j_h} \in D(X_j)$ is found, then (v_{i_l} v_{j_h} *y*) is added to the list of tuple supports *TupleSupport*^{A_i}, i.e. *y*=0 (*resp. y*=1), if $v_{j_h} \in D(X_j)$ (*resp.* $v_{i_l} \in D(X_i)$) is the *first* support of $v_{i_l} \in D(X_i)$ (*resp.* $v_{j_h} \in D(X_j)$). We must note that v_{j_h} is the first value support for v_{i_l} but they are also values support for each other by applying the bidirectionality property of relations associated to constraints.

A value v_{i_l} is deleted from $D(X_i)$ if and only if it has no viable value support. Each obtained set of deleted values for a variable should be announced immediately to the concerned acquaintances in order to save fruitless consistency checks for these values by the other agents. Obviously, reducing domains on an agent may cause an eventual domains' reductions on another agents. The same process, domains' reduction and exchange of deleted

values, should resumes until the full global arc-consistency is achieved or a domain wipes out, i.e. the problem is then detected as inconsistent.

Hence, all the agents starts the second step in order to prune more non-viable values. Each agent A_i checks first if it belong to a path formed by three variables. This is can be done by checking its list of constraint acquaintances, Γ^{A_i}. The same agent may belong to more than one path. First for each path, each agent A_i asks its path acquaintance agents[3] (A_k and A_j) for their sets of first support (*TupleSupport*^{A_k} and *TupleSupport*^{A_j}) with $\{X_i, X_j\}$ maintained by A_i, $\{X_i, X_k\}$ maintained by A_k, and $\{X_k, X_j\}$ maintained by A_j.

For each received set, the agent A_i determines first the boolean matrices M_{ik} and M_{kj} corresponding to the received *TupleSupport*^{A_j} and *TupleSupport*^{A_k} respectively. Second, performs the multiplication of these two matrices. Each entry of the obtained matrix $M_{Prod_{ij}}$ indicates the existence (entry equal to 1) or not (entry equal to 0) of a path of length 2 between the two variables X_i and X_j of the agent A_i through the variable X_k. Finally the agent performs the convolution of $M_{Prod_{ij}}$ and its first support matrix M_{ij} by applying the multiplication operator as follows:

$$\forall\, m \in \{1, \ldots, D(X_i)\} \text{ and } \forall\, l \in \{1, \ldots, D(X_j)\}$$
$$M_{Res}[m][l]=M_{Prod_{ij}}[m][l] * M_{ij}[m][l].$$

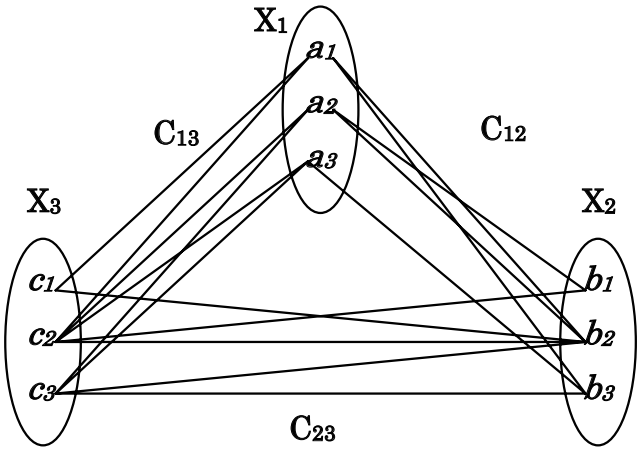


Figure 6.1. Example of arc-consistent problem.

To illustrate the principle of the proposed protocol, let us consider the example in Figure6.1 formed by three variables (X_1, X_2 and X_3) and its corresponding graph of first support in figure6.2. Figure6.3 shows the proposed model corresponding to the above example. Let us consider the agent A_1 responsible of the constraint C_{13} (*TupleSupport*^{A_1}=$\{(a_1\ c_1\ 0)\ (a_2\ c_2\ 0)\ (a_3\ c_2\ 0)\ (a_2\ c_3\ 1)\}$), it will receive the set of first support from its two acquaintances

[3]All the constraint agents belonging to the same path.

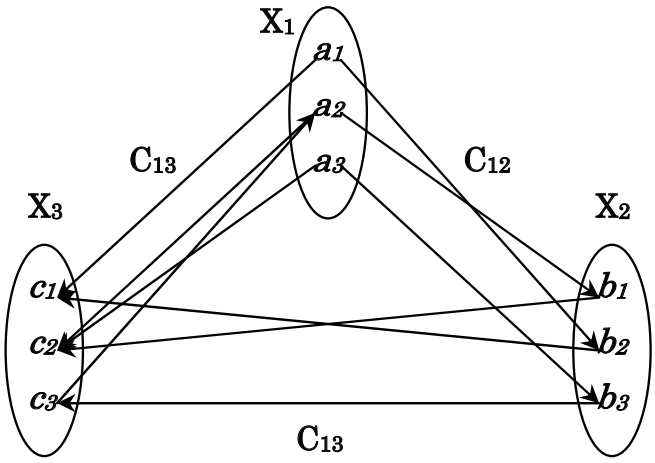


Figure 6.2. The corresponding graph of first support values.

(*TupleSupport*^{A_2}=$\{(a_1\ b_2\ 0)\ (a_2\ b_1\ 0)\ (a_3\ b_3\ 0)\}$ and *TupleSupport*^{A_3}=$\{(b_1\ c_2\ 0)\ (b_2\ c_1\ 0)\ (b_3\ c_3\ 0)\}$). It will determine the corresponding following matrices:

$$M_{12} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}, M_{23} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \text{ and } M_{13} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

Then, it will settle the product of the two first matrices:

$$M_{12}{*}M_{23}{=}M_{Prod_{13}}{=}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Finally the agent should determine M_{Res} using $M_{Prod_{13}}$ and M_{13} as mentioned above.

$$M_{Res} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

For each $a \in$ D(X$_i$) (*resp.* $b \in$ D(X$_j$))

If $\Sigma_{h=1}^{|D(X_j)|} M_{Res}[a][b_h]<1$ (*resp.* $\Sigma_{h=1}^{|D(X_i)|} M_{Res}[a_h][b]<1$)

then the agent should check if the value $a \in$ D(X$_i$) (*resp.* $b \in$ D(X$_j$)) is restricted path inconsistent or not and this by using the third criterion of the Property3.

Each value that does not satisfy the property conditions should be deleted and consequently propagated. For our example, we have to check only a_3 and c_3.

The same process is repeated for the other paths. However, enforcing local RPC on an agent may lead to AC enforcement, which in its turn leads to more RPC enforcement. Thus the same process should continue until the stable equilibrium state is reached. This state can

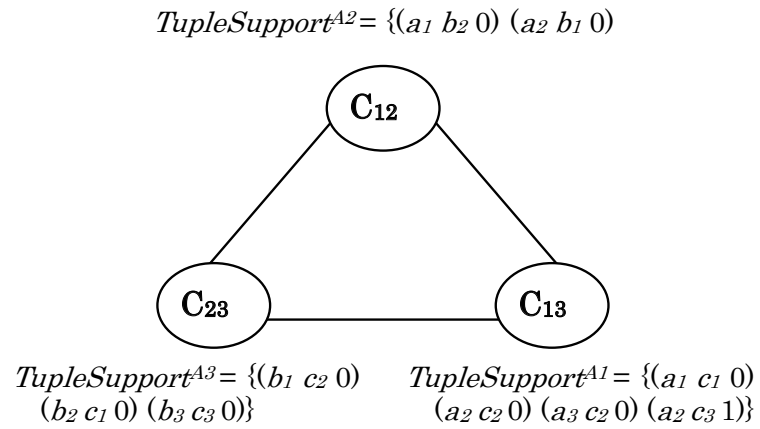


Figure 6.3. The corresponding model for the proposed approach

be defined by the satisfaction of all the agents of the system. An agent is satisfied if and only if it has no arc inconsistent or restricted path inconsistent value. It is noticeable that we can be content with enforcing *Lazy* RPC, one pass of RPC, in order to reduce the complexity of the pruning process.

Note that this dynamic allows a premature detection of failure: absence of solutions. Thus, in the case of failure, the constraint (which has detected this failure) sends a message to the interface in order to stop the whole process. For thus, the Interface agent in turn send a message to each constraint to ask them to stop their activities, and informs the user of the absence of solutions. The maximal reinforcement of global restricted path-consistency is obtained as a side effect from the interactions described above.

6.2 Discussion

6.2.1 Termination

The global dynamic of DRAC^{++} approach stops when the system reaches its finite stable equilibrium state. The state where all the restricted path inconsistent values are pruned or when one of the domains wipes out. At this state, all the agents are satisfied. However, in this second case, the problem is inconsistent i.e. no instantiation satisfies all the constraints.

The detection of the stable equilibrium state in a distributed system can be achieved by taking a snapshot of the system, using the well known algorithm of [19]. Termination occurs when all the agents are waiting for a message and there are no messages in the transmission channels. The cost, of the termination process, can be mitigated by combining snapshot messages with our protocol messages.

6.2.2 Complexity

Let us consider a CSP P having *n* for the total number of variables, *d* for the size of the variable domains and *e* for the total number of constraints. The number of Agents is *e*. If we consider a fully connected constraint network, we will have *e-1* acquaintances for each Constraint agent. DRAC^{++} is composed of two steps, the complexity of the first step is the same as DRAC approach. The space complexity of DRAC is (*n(n-1)/2*)*(*2d-1*) $\simeq$ O(n^2d), while its time complexity, in the worst case, is O(end^3).

For the second step, in the worst case, the maximal number of paths for each agent is (*n*-1)(*n*-2)/2$\simeq$ O(n^2). Each agent will receive the set of supports from its path acquaintance agents. it will first perform the boolean product of the two corresponding matrices with d^3 elementary operations (logical multiplication and logical additions). Second, it will perform the convolution of the obtained matrix with its set of supports. This process requires O(d^2) operations. Finally for each values of its second variable, the agent will check if it has a unique support, at least, in one variable of the two variables of the current path. This process requires O(d^3) operations. Thus the added process to DRAC approach requires, in the worst case, O(en^2d^3) operations, and O(ed^3) as additional space complexity.

6.3 Experimental comparative evaluation

In this section, we provide experimental tests on the performance and efficiency of the distributed filtering new approach DRAC^{++}. The experiments were performed over randomly generated instances using four parameters: *n* is the number of variables, *d* is the domain size of each variable, *p* is the graph connectivity (the proportion of constraint in the network, *p*=1 corresponds to the complete graph) and *q* is the constraint looseness (the proportion of allowed pairs of values in a constraint). The implementation was developed with Actalk [15] under Smalltalk-80 environment.

Two kinds of experiments where performed. The main goal of the first branch were dedicated to evaluate the efficiency of performing more than arc consistency for hard distributed constraint problems. Therefore, we have randomly generated a list of instances according to the following parameters, *n*=20; *d*=10 and $\langle p, q\rangle$ belonging to the transition phase, i.e., the most hard instances including arc-consistent and inconsistent problems, $\langle p, q\rangle$ = {0.2/0.3; 0.3/0.35; 0.4/0.35; 0.5/0.4; 0.6/0.4; 0.7/0.4; 0.8/0.42; 0.9/0.43}.

We have carried our experiments only on the most hard binary arc-consistent problems for DRAC and DRAC^{++}. The main goal is to highlight the usefulness of using meta-knowledge inferred from the set of first support to prune more inconsistent values on hard CN and with the minimum amount of additional constraint checks and CPU time. For each $\langle p, q\rangle$, 70 CNs instances were randomly generated (the total number of generated instances is 560) and processed using both approaches DRAC and DRAC^{++}. Note that regarding DRAC^{++} we performed only *lazy* RPC in order to show that for some problems only partial RPC is

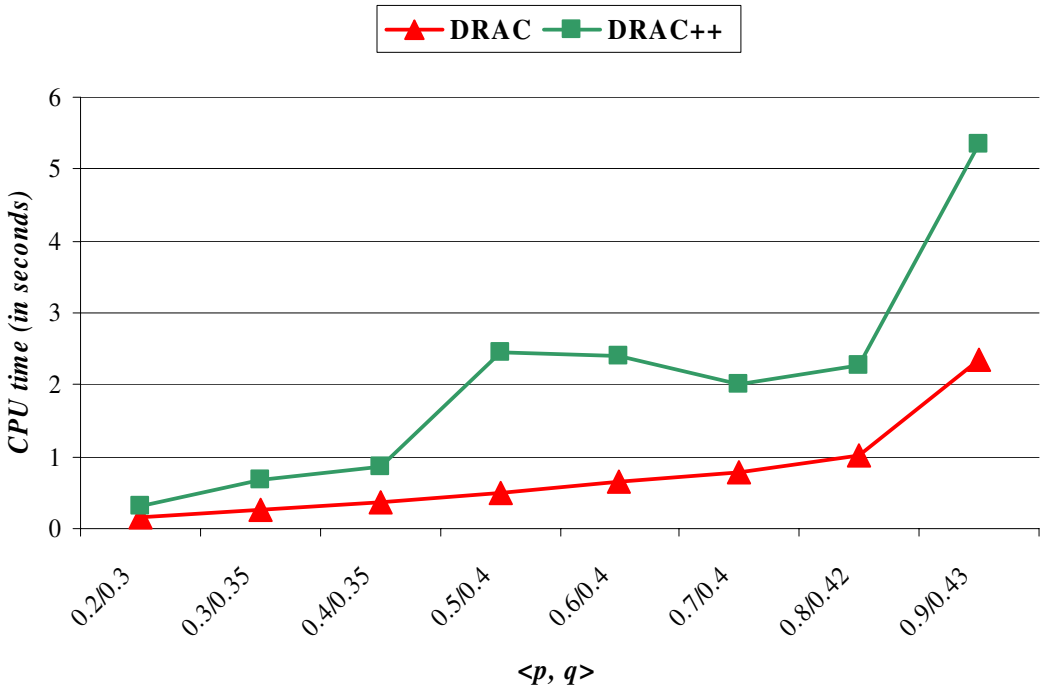


Figure 6.4. DRAC vs. DRAC^{++} mean results in term of the required CPU time for hard arc-consistent problems.

enough to prove the inconsistency especially of almost all over-constrained problems. Table 6.1 illustrates the percentage of arc-consistent instances among the 70 generated ones.

The results reported below represent the average of the obtained outcomes in means of four criteria, the CPU time in seconds, the percentage of deleted inconsistent values, the number of constraint checks and the number of exchanged messages for DRAC and DRAC^{++}.

Figures 6.4 and 6.5 show that performing partial RPC on arc-consistent hard problems allow us to discard more values (up to 7 times for $\langle$ 0.5; 0.4 $\rangle$) and especially to detect the inconsistency of a high proportion of them in a reasonable additional CPU time. For example, in Table 6.2. all the over-constrained arc-consistent problems are proved to be inconsistent for $\langle 0.4; 0.35\rangle$, $\langle 0.5; 0.4\rangle$; $\langle 0.6; 0.4\rangle$, $\langle 0.7; 0.4\rangle$, $\langle 0.8; 0.42\rangle$ and $\langle 0.9; 0.43\rangle$.

Table 6.1. Percentage of arc consistent instances among the 70 generated ones

	$\langle$**0.2; 0.3**$\rangle$	$\langle$**0.3; 0.35**$\rangle$	$\langle$**0.4; 0.35**$\rangle$	$\langle$**0.5; 0.4**$\rangle$
%Inconsistent Problems	77.14%	85.71%	37.14%	95.71%
	$\langle$**0.6; 0.4**$\rangle$	$\langle$**0.7; 0.4**$\rangle$	$\langle$**0.8; 0.42**$\rangle$	$\langle$**0.9; 0.43**$\rangle$
%Inconsistent Problems	67.71%	35.71%	64.21%	58.57%

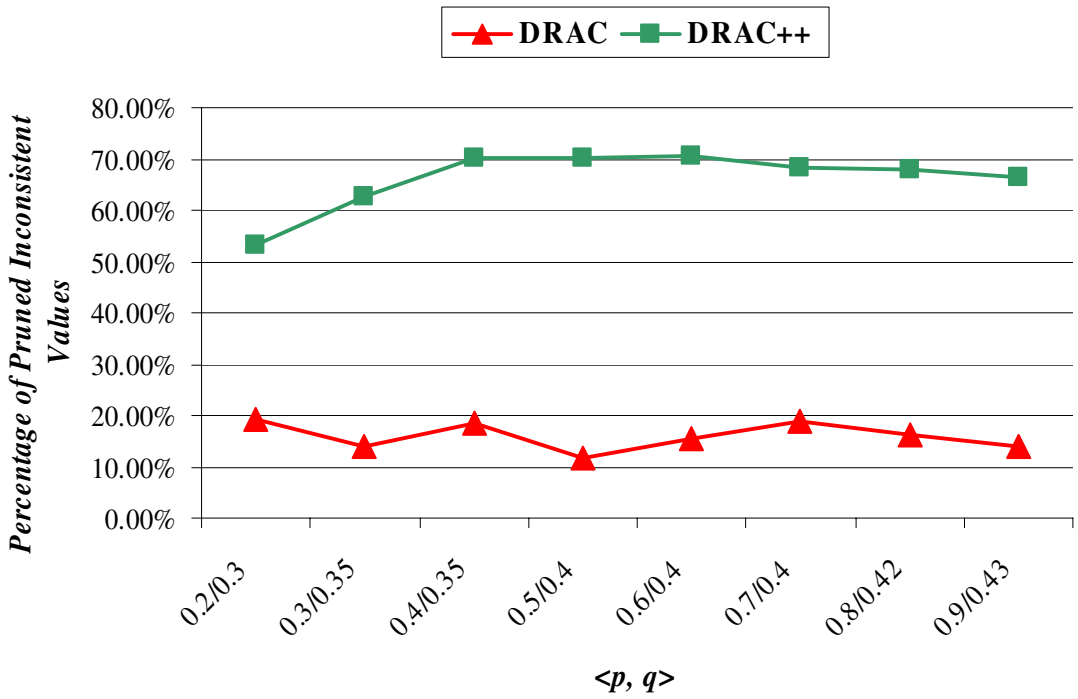


Figure 6.5. DRAC vs. DRAC^{++} mean results in term of the percentage of pruned inconsistent values. All tested instances are initially arc-consistent.

While for under-constraint problems $\langle 0.2; 0.3\rangle$, $\langle 0.3; 0.35\rangle$, the difference in the percentage of reduced values is lesser. As for the additional needed CPU time for DRAC^{++}, it varies from 0.15 seconds for sparse problems to 3 seconds for the most dense problems (case $\langle 0.9; 0.43\rangle$).

Figures 6.6. and 6.7. give the obtained results for the number of constraint checks (ccks). We can say that the new protocol requires only few supplementary ccks especially for the case of loose CN (cases $\langle 0.2; 0.3\rangle$, $\langle 0.3; 0.35\rangle$, and $\langle 0.4; 0.35\rangle$). However, for the cases $\langle 0.5; 0.4\rangle$ and $\langle 0.9; 0.43\rangle$ to prove the inconsistency necessitates greater ccks. Nevertheless, the true number of ccks needed for these instancesis much greater. The use of the collected knowledge of first support allows to decrease the amount of ccks and consequently to amend the efficiency of the pruning process. This claim will be approved in the next branch of experiments.

As regard with the exchanged number of messages, at first glance it seems that DRAC^{++} requires a large number of messages to reinforce RPC; this result can be vindicated by the fact that in the beginning of the second step, all the agents implied in at least one path should exchange their set of first support, so this may increase the amount of messages especially, for over-constrained problems.

At this point, we can say that performing "*even lazy*" restricted path consistency is worth-

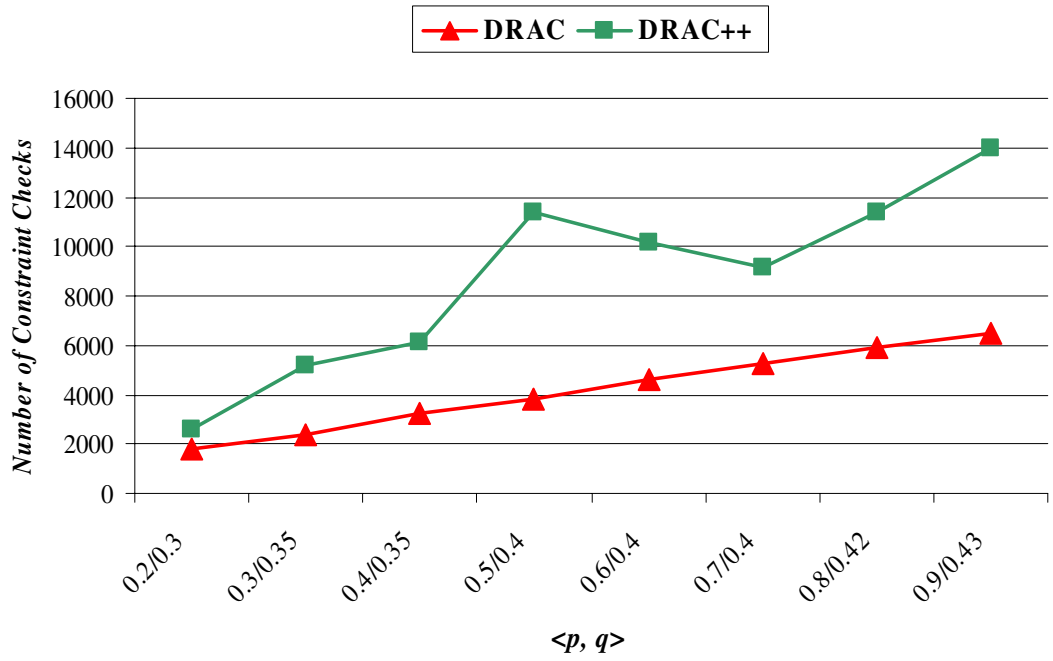


Figure 6.6. DRAC vs. DRAC^{++} mean results in term of the number of constraint checks for hard arc-consistent problems.

while especially, for over-constrained problems. This can be justified by the fact that for such problems, the probability of having a path of three variables in the CN is high compared to under-constraint problems leading to the discovery of more path inconsistent values and consequently to more reduction.

As for the second branch of experiment, We brought out two versions of DRAC^{++}: DRAC^{++}-1 without proposed property and DRAC^{++}-2 with the proposed property , respectively. The main objective of these experiment is to evaluate the performance of the proposed property using the same previous parameters. The results reported below represent the average of the obtained outcomes in terms of three criteria: the CPU time in seconds, the percentage of pruned values, and the number of constraint checks (ccks).

At first glance the result in Figure6.8 shows that DRAC^{++}-2 required little more CPU time ($\simeq$14%) than DRAC^{++}-1. This additional CPU time is used in order to decrease the number of constraint checks. Figure6.10 shows that the use of the proposed property leads to save almost 30% of the needed number of ccks. The saving of ccks increases hand-in-hand with the hardness of the problem.

The difference in the percentage of deleted values noticed between the two versions of DRAC^{++} (Figure6.9) is vindicated by the fact that the used instances include restricted path consistent instances and inconsistent instances. Therefore, the number of pruned values vary

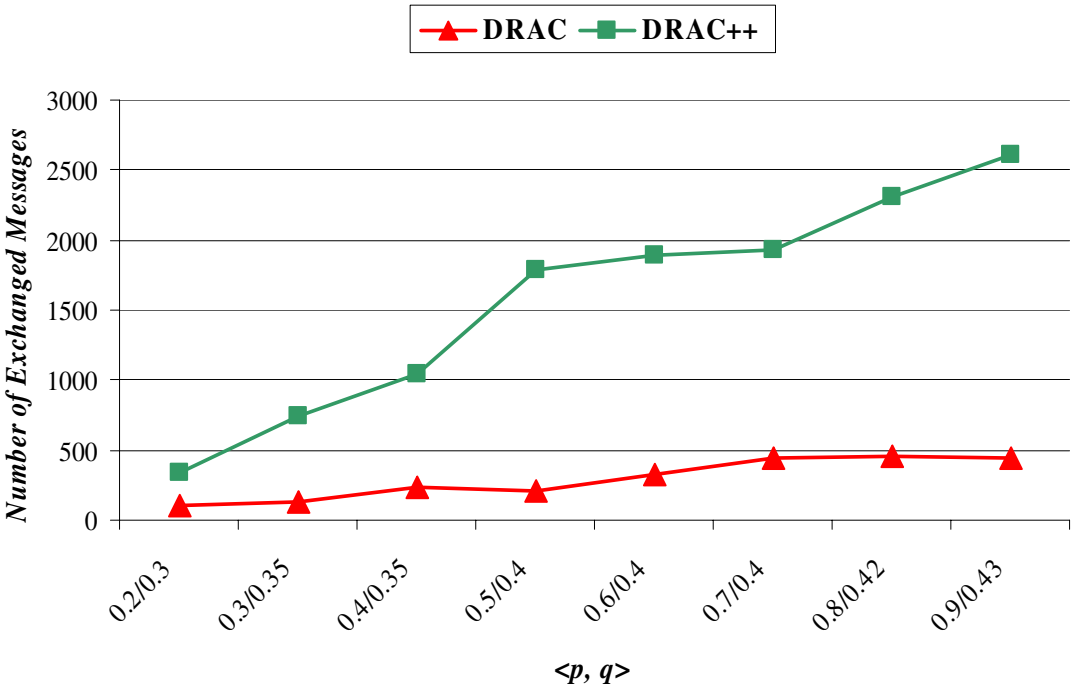


Figure 6.7. DRAC vs. DRAC^{++} mean results in term of the number of exchanged messages for hard arc-consistent problems.

Table 6.2. Percentage of problems detected as inconsistent among the arc-consistent problems

	⟨**0.2; 0.3**⟩	⟨**0.3; 0.35**⟩	⟨**0.4; 0.35**⟩	⟨**0.5; 0.4**⟩
%Inconsistent Problems	62.96%	81.66%	100%	100%
	⟨**0.6; 0.4**⟩	⟨**0.7; 0.4**⟩	⟨**0.8; 0.42**⟩	⟨**0.9; 0.43**⟩
%Inconsistent Problems	100%	100%	100%	100%

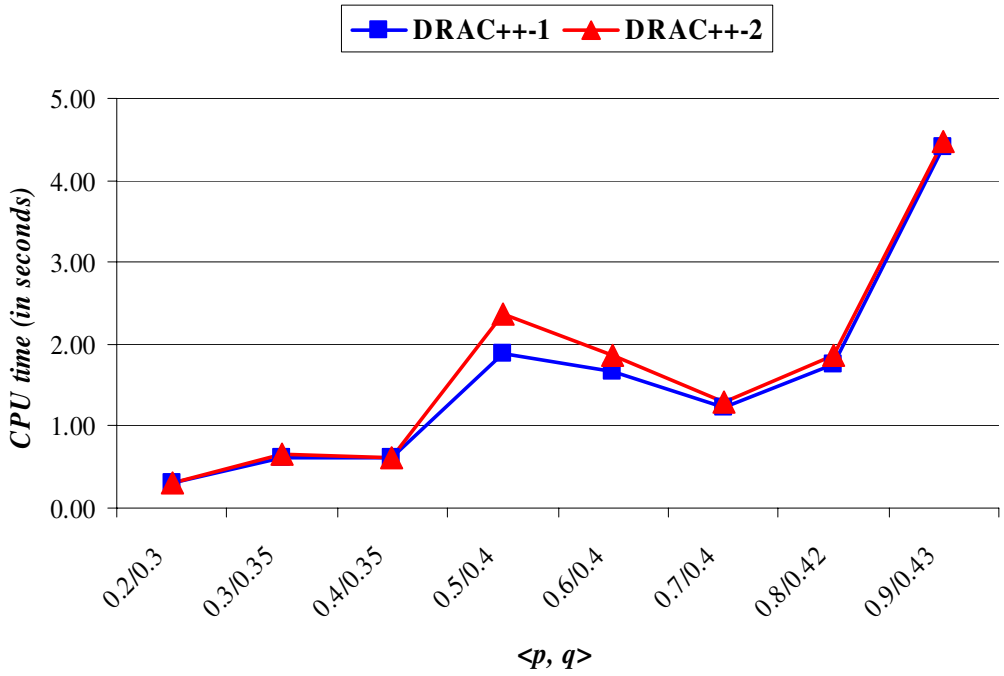


Figure 6.8. Results of DRAC^{++}-1 without the proposed property vs. DRAC^{++}-2 with the proposed property, in mean of CPU time.

for both approaches. Table6.3 shows that almost in all cases DRAC^{++}-2 prunes less values to prove the inconsistency of the instances. While for restricted path consistent instances, the two approaches prunes the same non-viable values.

We carried out hypothesis testing, dependant two samples t-test, on the above results found in terms of constraint checks for both approaches DRAC^{++}-1 without property and DRAC^{++}-2 with property. The goal is to statistically prove the accuracy of the above result. The formalization of both the null hypothesis and the alternative hypothesis is as follows:

$H_0 : \mu_{CckDRAC^{++}-2} = \mu_{CckDRAC^{++}-1}$
$H_1 : \mu_{CckDRAC^{++}-2} < \mu_{CckDRAC^{++}-1}$

The means of the 70 random samples are measured using Matlab6.1 using significance level $alpha$ = 0.05. Table 6.4 reports the obtained results for each pair $\langle p, q \rangle$. Regarding these results the null hypothesis H_0 is rejected in most cases with low significance varying from 0.0306 to 4.44E-06. The small significance indicates the strong rejection of the null hypothesis, which means that the result is highly statistical significant. However, for the cases $\langle 0.4, 0.35 \rangle$ and $\langle 0.7, 0.4 \rangle$, the null hypothesis is not rejected, which means that only in these two cases the means in term of constraint checks for both approaches is almost the

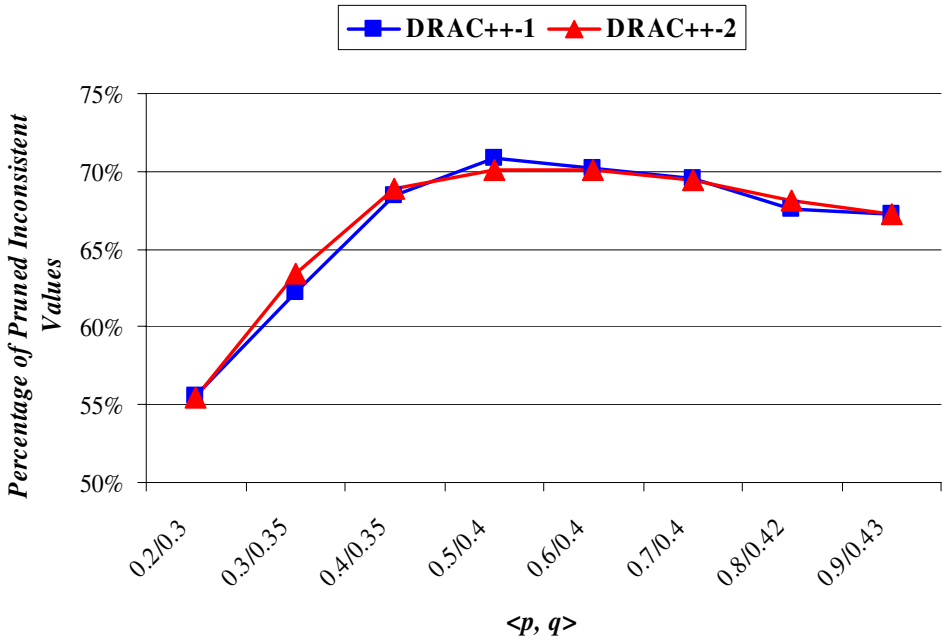


Figure 6.9. Results of DRAC^{++}-1 without the proposed property vs. DRAC^{++}-2 with the proposed property, in mean of percentage of deleted inconsistent values.

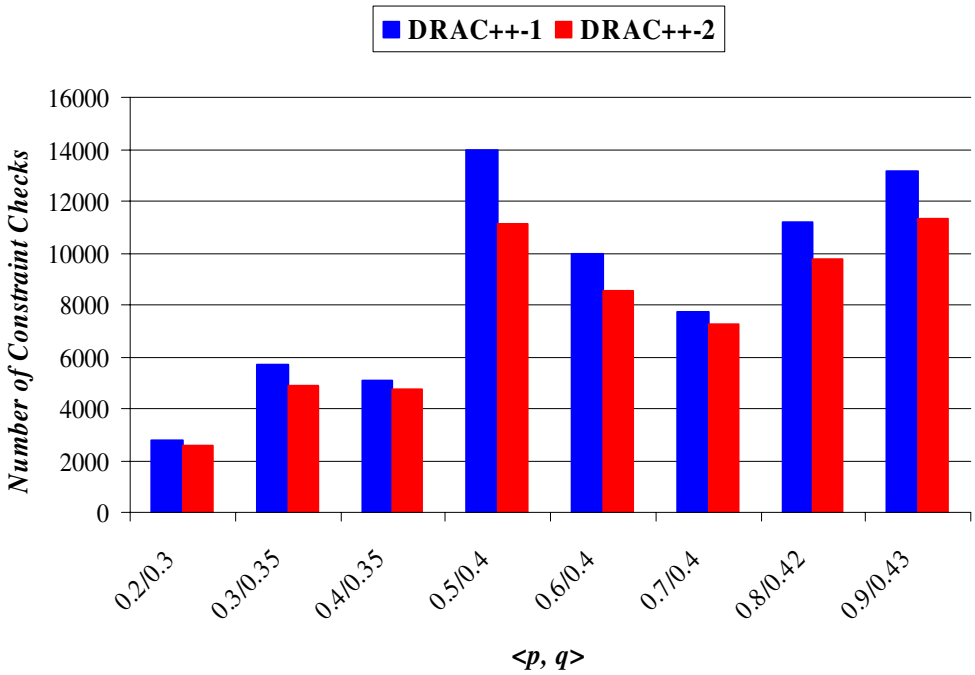


Figure 6.10. Results in terms of the mean of the required number of ccks

Table 6.3. Results of the percentage of deleted values for the inconsistent instances.

	$\langle$**0.2; 0.3**$\rangle$	$\langle$**0.3; 0.35**$\rangle$	$\langle$**0.4; 0.35**$\rangle$	$\langle$**0.5; 0.4**$\rangle$
DRAC^{++}-1	66.50%	68.70%	68.09%	61.00%
DRAC^{++}-2	63.41%	68.7%	68.09%	71.00%
	$\langle$**0.6; 0.4**$\rangle$	$\langle$**0.7; 0.4**$\rangle$	$\langle$**0.8; 0.42**$\rangle$	$\langle$**0.9; 0.43**$\rangle$
DRAC^{++}-1	62.42%	70.13%	68.26%	68.71%
DRAC^{++}-2	69.42%	70.24%	68.26%	68.71%

Table 6.4. Dependent two samples t-test for the number of constraint checks means of both approaches DRAC$^{++}-1$ and DRAC$^{++}-2$, for each pair $\langle p, q\rangle$.

	Decision	**Significance**	**Confidence Interval**
0.2/0.3	*reject* H_0	0.0147	]-Inf, -50.9654]
0.3/0.35	*reject* H_0	3.03E-05	]-Inf, -485.9168]
0.4/0.35	*accept* H_0	0.0803	]-Inf, 64.4363]
0.5/0.4	*reject* H_0	4.44E-06	]1.0e+003* -Inf, -1.8078]
0.6/0.4	*reject* H_0	0.0158	]-Inf, -336.4257]
0.7/0.4	*accept* H_0	0.0819	]-Inf, 90.6016]
0.8/0.42	*reject* H_0	0.011	]-Inf, -405.7068]
0.9/0.43	*reject* H_0	0.0306	]-Inf, -224.9553]

same. The significance for both cases is around 0.0800 which means that we would have observed values of T more extreme that the one in this sample in 800 among 10000 similar experiments.

6.4 Summary

The main objective of this chapter is to achieve full global restricted path consistency (RPC), for any binary constraint network, in an entirely distributed way without any help from centralized algorithms. Therefore, we described a novel agent-based approach DRAC^{++} which is a continuity of the work presented in Chapter5 in which we prune substantially more non-viable values than DRAC approach with the minimal amount of constraint checks and reasonable CPU time.

The termination and complexity of the new protocol have been proved. The experimental comparative evaluation shows that this approach is worthwhile especially for hard over-constrained problems. The new approach is based on DRAC approach and have been

published in [12, 13, 14, 15] and under reviewing in the *International Journal of Artificial Intelligence Tools* [1].

Algorithm 3 Start message executed by each Constraint agent A_i

begin

1: **for all** path (X_i, X_j and X_k) such that X_i, $X_j \in$ *Var*($C_{ij}^{A_i}$) **do**
2: /∗Assume that the problem is initially arc consistent and the agent A_i has received *TupleSupport*^{A_j} and *TupleSupport*^{A_k} from its two Constraint path acquaintances A_j and A_k maintaining $C_{kj}^{A_j}$ and $C_{ik}^{A_k}$ respectively/∗
3: Create M_{ik} and M_{kj} corresponding to *TupleSupport*^{A_k} and *TupleSupport*^{A_j};
4: Create M_{ij} corresponding to its $SP_{X_iX_j}$;
5: Compute M_{Res}= M_{ik} * M_{kj};
6: Perform the convolution of M_{Res} and M_{ij} as defined above;
7: **end for**
8: **for all** $X_i \in$ *Var*($C_{ij}^{A_i}$) **do**
9: **for all** $v_{i_l} \in D^{A_i}(X_i)$ **do**
10: **if** $\sum_{l\in[1..|D^{A_i}(X_j)|]} M_{Res}[v_{i_l}][l] < 1$ **then**
11: **if** ((*EnforceRPC*:v_{i_l} *for*: X_i)= false) **then**
12: addTo(*IncValue*^{A_i}[X_i], v_{i_l});
13: **end if**
14: **end if**
15: **end for**
16: **end for**
17: **for all** $v_{i_l} \in$ *IncValue*^{A_i}[X_i] **do**
18: delete($D^{A_i}(X_i)$, v_{i_l});
19: **if** $D^{A_i}(X_i)=\emptyset$ **then**
20: Send(*Self*, Interface, "*StopBehavior*");
21: **end if**
22: **end for**
23: **for all** $A_j \in$ *AcqConst*^{A_i}(X_i) **do**
24: *Send*(A_j, Self, "*UpdateDomain*: *IncValue*^{A_i}[X_i] *for*: X_i");
25: **end for**

Algorithm 4 *EnforceRPC: for:* message executed by each invoked agent A_i

EnforceRPC: *a* **for**: X_i

```
1: total ← 0;
2: for all t ∈ TupleSupport^{A_i} do
3:    if t[i] > a and t[3]=(2-i) then
4:       total ← total+1;
5:    end if
6:    if t[i] = a then
7:       nbsupport ← nbSupport+1;
8:       uniqSup ← t[j];
9:    end if
10: end for
11: if nbSupport ≤ 1 then
12:    if total < (|D^{A_i}(X_j)| -1) then
13:       if ((CheckOneSupport:X_i for: a from: uniqSup) = true) then
14:          Determine set1 set of support of X_i=a in X_k;
15:          Determine set2 set of support of X_j = uniqSup in X_k;
16:          if (set1 ∩ set2 = ∅) then
17:             c ← smallest support of a;
18:             found ← false;
19:             while found do
20:                c' ← searchNextSupport:uniqSup of:X_j in:X_k;
21:                if (a, c') satisfies C_ik then
22:                   found ← true;
23:                   return false;
24:                end if
25:             end while
26:          end if
27:       end if
28:    end if
29: end if
30: return true;
```

Chapter 7

Taking DRAC to the Real World: An Efficient Complete Solution for Static Meeting Scheduling

In the previous chapter we introduced a new agent-based approach to enforce AC on any CN. The obtained results motivated us to take this approach to a real world application, meeting scheduling (MS) problem. As we mentioned before, this problem is still attracting the attention of many researchers due especially to its preeminent role in the success of many organizations. However, DRAC cannot solve a problem *P* but more precisely, allow to produce a new instance *P'* more simple and equivalent, i.e., having same set of solutions.

In this chapter we propose a novel complete approach to solve any MS problem with predictable structure, i.e., a problem where the set of coming meetings is known beforehand and fixed. This approach use DRAC protocol during search for solutions in order to make the search easier and also to detect earlier global inconsistency. In the following we introduce first our proposed formalization for any static MS problem. Second, we describe how DRAC model can be adapted to our MS solver followed by a detailed description of the global scenario for static MS solver. Third we discuss the termination and complexity properties. Then, we present the experimental results. Finally we summarize this chapter.

7.1 Formalization for any static meeting scheduling problem

We propose to formalize any static MS problem as a VCSP (valued constraint satisfaction problem) [88].

Definition 38 *We define a static Meeting Scheduling problem, as a valued constraint satisfaction problem (VCSP) quintuples (X, D, C, S, φ) where*

- *X = $\{X_1, \ldots, X_n\}$ is the set of **n** meetings that need to be scheduled. X_k with $k \in \{1, \ldots\ n\}$ denotes the k^{th} meeting to schedule.*

- *D = $\{D_1, \ldots, D_n\}$ is the set of all possible time slots for all the meetings X. D_i=$\{dt_{i_1}$, $\ldots$, $dt_{i_d}\}$ (with $|D_i|$=d), is the set of possible time slots for the meeting X_i.*

- *C is the set of all constraints of the problem. We divide the set C into two types of constraints: constraints related to the users and constraints related to the meetings. For the former, we can consider:*

  - *hard constraints: C_h related to the non-availability of all users.*
  - *soft constraints, C_s related to the preferences of all users towards the possible dates in their calendar.*

  With regard to the second type of constraints, $C_{allDiff}$, it represents the set of allDifferent constraints relating each pair of meetings X_k and X_l sharing at least one participant A_j ($A_j \in$ Part(X_k) and $A_j \in$ Part(X_l)[1].).

- *φ : C $\rightarrow$ E. C=$\{C_h \cup C_{allDiff} \cup C_s\}$, for each hard constraint $c_i \in \{C_h \cup C_{allDiff}\}$ we associate a weight $\perp$ and for each soft constraint $c_j \in C_s$ we associate a weight $w_j \in$ [0..1][2]. This weight reports the degree of preference of a user to have a meeting at the date dt_j.*

- S *represents the valuation structure that defines the proposed optimality criteria (discussed in next section) and will be used to find the "best" solution.*

In addition, for each meeting X_k we assign a different weight $W_{X_k} \in$ [0..1] to define the degree of importance of X_k ($k \in \{1, \ldots, n\}$) and it is used to allow the processing of the most important *meeting*[3] *at first.*

Solving a MS problem consists in finding a "good" assignment *sl*$^* \in$ *Sol*:=$D_1 \times \ldots \times D_n$ of the variables in *X*=$\{X_1, \ldots, X_n\}$ according to their importance W_{X_k}, such that all the hard constraints are satisfied while maximizing the utility of the Proposer agents (*selfish* protocol). The *GU* is defined by the summation of the preferences of all the attendees for all the scheduled meetings such that:

$$sl^* = arg\ max_{sl \in Sol} GU(sl) \tag{7.1}$$

$$GU = \sum_{k \in \{1,\ldots,|sl|\}} w_k^{A_i} \tag{7.2}$$

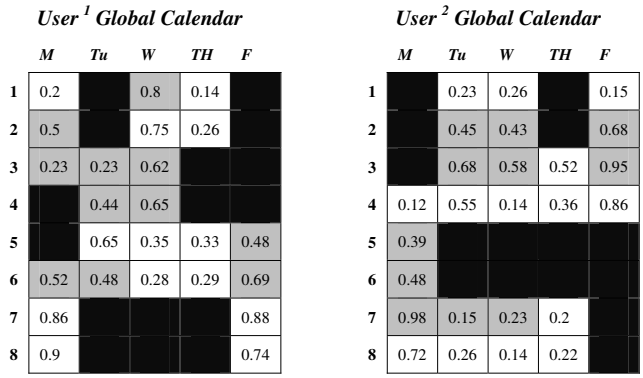


Figure 7.1. Example of a user calendar consisting of non-availability of the user (black boxes), the possible time slots for the current meetings (gray boxes) and the favorite time slots with their corresponding degree of preferences.

To illustrate this formalization more clearly, let us consider the following example consisting of 2 users, each entrusted with the task of scheduling one meeting. Assume that both meetings require the participation of all the users. Figure 7.1 illustrates the preferences of each user. The underlying MS formalization (X, D, C, S, φ) is as follows:

- X=$\{X_1, X_2\}$,
- D=$\{D_1, D_2\}$, D_k is represented by the gray boxes in Figure 7.1, i.e., possible time slots for the underlying meeting.
- C= $C_H \cup C_S$ where:
  - C_H is represented by both the black boxes in Figure 7.1 and all the *allDiff* constraints existing between each pair of meetings ($X_1 \neq X_2$).
  - C_S represented by the clear boxes in Figure 7.1. The white boxes represent the favorite time slots while the gray boxes represent the possible time slots for the current meeting to schedule. The number inside the boxes indicates the degree of preferences w_k of each user for each time slot in their calendar.

7.2 DRAC model adapted to the MS problem

Lets recall that The DRAC model uses two kinds of agents:

[1]The function *Part*(X_k) denotes all the participants in the meeting X_k

[2]This assumption does not contradict the ability of our protocol to support any kind of preferences' measurement evaluation.

[3]We assume that the users report truly and accurately the importance of their meetings.

- Constraint agents
- Interface agent

Each agent has its own knowledge (static and dynamic), a local behavior to satisfy, and a mailbox to store incoming messages. The agents communicate by exchanging asynchronous point-to-point messages. An agent can send a message to another only if it knows the other belongs to its acquaintances. For transmission between agents, we assume that the messages are received in a finite delivery time and in the same order they are sent. Messages sent from several agents to a single one may be received in any order. The Interface agent is an intermediate interface between all the Constraint agents. It is added in order to create the agents and, most importantly, to inform the users of the result.

This model can be ”well” adapted to the MS problem. In this problem, each Constraint agent can be considered as a User agent A_i acting on behalf of a human user. A User agent must maintain the concerned human user's calendar for his/her availability, preferences and the already planned meetings. The acquaintances of an agent consist of all of the agents that should be present in the same meeting, called Participant agents (represented as *Part*(X_k)). Accordingly, in our system an agent is considered as a Proposer agent when it has a meeting to schedule. It can be also considered as a Participant agent if it is a participant in another meeting proposed by another agent of the system.

Each scheduled meeting that has been registered (represented as $Calendar^{A_i}$) is considered as a new constraint. Therefore it must be added to the set of hard constraints maintained by the corresponding agents. Each agent A_i maintains a VCSP^{A_i} for which the variables $X^{A_i} \in X$ represent the meetings dates to found for its user's set of meetings (represented as $Meetings^{A_i}$), while the constraints $C^{A_i} \in C$ (C^{A_i}=$C_H^{A_i} \cup C_S^{A_i} \cup C_{allDiff}^{A_i}$) represent the non-availability, the preferences, the timetabling of the corresponding user and the constraints relating to each pair of its meetings.

Thus in the proposed model, the aforementioned constraints represent the intra-agent constraints for A^i while the inter-agent constraints are represented by a set of strong constraints, i.e., equality constraints. An equality constraint exists between agents A_i and A_j if and only if *at least* one meeting $X_k^{A_i}$ (*resp.* $X_h^{A_j}$) exists, such that $A_j \in Part(X_k^{A_i})$ (*resp.* $A_i \in Part(X_h^{A_j})$). It is noteworthy that the inter-agent constraints are dynamic because the participants and their number in a meeting differ from one meeting to another. Each attendant has a set of meeting preferences for each particular meeting. The local goal is to schedule meetings such that all its hard constraints C_H are satisfied while trying to maximize the *GU*. The global goal is to schedule the maximum of users meetings satisfying all the inter-agent constraints.

In this approach, we consider the standpoint of the host of each meeting (who can be the director of the company or the manager of the department, etc.). In our scenario, we will adopt the natural, innately fair and self centered behavior of a human being; since the knowledge of a user is self centered knowledge. Hence, each agent in the meeting scheduling

process tries to satisfy its local goal while maximizing its preferences (*selfish* protocol). The adopted criteria for an MS solver should guarantee some common attributes for both the resulting decision and the scheduling process itself. Furthermore, in order to ensure such features for the solver and the outcomes, the proposed system should be able to extract the truthful preferences [35] and availabilities of the users.

Hence, the optimal solution is based on self-centered initiator preferences. Overbidding the preferences for any time cannot change the outcome; therefore the dominant strategy for every agent is to reveal its utility values truthfully. Obviously, the optimality criteria may differ from one scenario to another according to the measures adopted by the system designer, e.g., a pure utilitarian approach [35], the Nash approach [75], and others, while, the global proposed dynamic remains the same for any chosen measurement. In the case of a global measurement, e.g., maximizing the summation of the participants' utilities, we propose as stated in the work of [35] to use convenience points to express preferences over alternative times for every proposed new meeting. In addition, we propose to embed the Clarck Tax mechanism [34] to incite the users to truly express their preferences towards the meeting's importance and users' possible timing. As for the non-availability of the participant, we assume that the users will reveal their real availability if necessary.

Hence, each participant in a meeting will get a fixed amount of convenience points to spread among its availability-times according to its preferences and this for each new meeting added to the system. The Tax can be computed on the amount of convenience points given for the next meeting. In our formalization, the weights $w_k^{A_i}$ associated to soft constraints will be dynamic and differs according to the current meeting to schedule. The sum of weights should be equal to the current amount of convenience points.

7.3 Global scenario for static MS solver

The global objective of the proposed approach is to schedule all meetings for all of the users while maximizing their local preferences. In addition, we focused on minimizing the total amount of exchanged messages. The multi-agent meeting scheduling negotiation protocol is divided into two steps:

- The first step uses the basic idea of the DRAC approach, which consists in transforming the original MS problem into another equivalent MS'. This step is needed to reinforce some level of local consistency [65] (node and arc consistency) in the initial problem.

- The second step solves the obtained MS problem while maintaining arc-consistency and this is accomplished via interactions and negotiations between Participant agents and the Proposer agent. Each Proposer agent searches for the best solution for its meetings that, on the one hand, fulfils the condition given in the previous section, and on the other, satisfies all hard constraints.

When a user wants to host a meeting, he has to run the Interface agent, which will activate the corresponding Proposer agent and make it interact with all of the Participant agents. More than one Proposer agent can be activated at the same time, i.e., in the case of multiple users who want to schedule their meetings.

Algorithm 5 Start message executed by each Proposer agent A_i.

begin

1: **for all** $X_k^{A_i} \in Meetings^{A_i}$ **do**
2: **for all** $dt_{k_l} \in D_k^{A_i}$ such that $dt_{k_l} \in C_{A_i}^H$ OR $\exists\, X_h^{A_j}$ / $X_h^{A_j} \in Calendar^{A_i}$ and $X_h^{A_j} = dt_{k_l}$ **do**
3: $D_k^{A_i} \leftarrow D_k^{A_i} \; dt_{k_l}$;
4: **end for**
5: **end for**
6: **if** ($D_k^{A_i}$=$\emptyset$) **then**
7: change calendar $D_k^{A_i}$ of $X_k^{A_i}$;
8: **else**
9: **for all** $A_j \in Part(X_k^{A_i})$ **do**
10: Send(A_j,*self*, "*ReduceCalendar*:$D_k^{A_i}$ *for*:$X_k^{A_i}$");
11: **end for**
12: **end if**

Each activated Proposer agent must first reduce the time slots of the corresponding meetings according to its hard constraints, constraints defining the non-availability of the user. This process can be viewed as a local reinforcement of node consistency and aims to reduce the meetings' slot times by eliminating the dates upon which the meeting cannot be held. In other words, a meeting cannot be held on a date defined as a non-available date for the user or already planned for another meeting.

If the time slots for a meeting become empty after reduction that indicates that the corresponding user is not available for all of the proposed dates of this meeting. The time slots of this meeting must then be changed. Otherwise, the Proposer agent must send the obtained reduced time slots for all of the current meetings to be scheduled to all of the Participant agents. Each Participant agent that receives this message starts first by eliminating both the non-viable dates from the received time slots of the meetings (dates that correspond to its non-availability), and all the dates taken by already scheduled meetings. After that, it returns the obtained time slots to the sender agent.

At first, the Proposer agent collects all the received reduced slot times, then, begins by scheduling its meetings. It tries to first find the proposal that maximizes its preferences and then sends it to the concerned acquaintances. If the Proposer agent cannot find a solution to this problem, it changes the time slot of this meeting. Each agent, that receives this proposal,

Algorithm 6 Main procedures executed by each Proposer agent A_i

ReduceCalendar:D ***for***:m

1: **for all** $d \in D$ such that $d \in C_H^{A_i}$ OR $X_h^{A_j}$ / $X_h^{A_j} \in Calendar^{A_i}$ and $X_h^{A_j} = d$ **do**
2: $D \leftarrow D \setminus d$;
3: **end for**
4: *Send*(*Sender*, *self*, "*Reply*:*D for*:$X_h^{A_j}$");

Reply:D **for**:$X_k^{A_i}$

1: *SetD*$\leftarrow$ *SetD* $\cup$ *D*;
2: **if** (Size(*SetD*)=|*Part*($X_k^{A_i}$)|) **then**
3: $D \leftarrow D \cap SetD_{i \in \{1,\ldots,|SetD|\}}[i]$;
4: **end if**
5: **if** ($D = \varnothing$) **then**
6: Change $X_k^{A_i}$ possible times;
7: **else**
8: Choose $d \in D$ / satisfy Eq.(1);
9: **for all** $A_j \in$ *Part*($X_k^{A_i}$) **do**
10: *Send*(A_j, $self$,"*ReceiveProposal*:d *for*:$X_k^{A_i}$");
11: **end for**
12: **end if**

must first check if it has, meanwhile, accepted another proposal for the same date. In the negative case, the agent will first update its hard constraints by adding the new proposal, then update the dates of its not-yet-scheduled-meetings by eliminating the dates that correspond to the same date of the just scheduled meeting, in order to maintain the arc-consistency. Finally it informs the Proposer agent of its agreement. However, if the agent has another meeting already scheduled at the same time as the proposed meeting, it must send a negative answer to the Proposer agent and ask it to change its proposal.

Accordingly, each agent that has proposed a meeting and received at least one negative answer must change its proposal. Consequently, this agent must decrease its degree of preferences and the same process is repeated until an agreement is reached among all of the participants. If after testing all of the solutions no agreement is reached, the Proposer agent is obliged to inform the participants of the meeting cancelation.

The aforementioned dynamic resumes running until the system reached its stable equilibrium state. This state can be defined as the satisfaction of all agents in the system. The satisfaction of an agent is defined as the scheduling of all its meetings or the cancelation of the ones that cannot be held at that time.

We should emphasize the fact that in this paper we assume on the one hand that each newly scheduled meeting will be considered as a hard constraint, and on the other hand, each

Algorithm 7 Main procedures executed by each Proposer agent A_i.

ReceiveProposal:*d* **for**:*X*

1: res ← true;
2: **if** ($\exists\ X_h^{A_j} \in Calendar^{A_i}$ and $X_h^{A_j}$ = *d*) **then**
3: res ← false;
4: **end if**
5: **if** (res=true) **then**
6: Add($Calendar^{A_i}$, ($X_h^{A_j}$, d));
7: Update set C_H;
8: **end if**
9: *Send*(*Sender*, *self*, "*Response*:res *for*:*X*");

Response:res **for**:*X*

1: *setRep* ← *setRep* ∩ res;
2: **if** (Size(*SetRep*)=|*Part*($X_k^{A_i}$)|) **then**
3: **if** (*SetRep*[*i*],*i* ∈ {1..|$SetRep$|}/ *SetRep*[*i*]=false) **then**
4: Choose another date *d*';
5: **for all** $A_j \in$ *Part*($X_k^{A_i}$)$ **do**
6: *Send*(A_j, *self*, "*ReceiveProposal*:*d*' *for*:$X_k^{A_i}$");
7: **end for**
8: **else**
9: **for all** $A_j \in$ *Part*($X_k^{A_i}$) **do**
10: *Send*(A_j, *self*, "*Confirmation*:d *for*:$X_k^{A_i}$");
11: **end for**
12: **end if**
13: **end if**

agent performs a selfish protocol. This choice is used in order to avoid dynamic changes and especially to escape from an infinite processing loop. This work can be considered as the first version of the proposed approach. The integration of the dynamic process will be discussed in the next chapter where the proposed protocol focused on maximizing the utility of all the agents of the system.

7.4 Theoretical discussion

7.4.1 Termination detection

The dynamic of the MSRAC approach ends when the system reaches its stable equilibrium state. In real application, this state will be temporary, and the whole system will restart with

new sets of meetings to schedule. However, at the stable equilibrium state all the agents are satisfied; that satisfaction is defined for each agent by two aspects, the completion of scheduling all its current meetings, and the acquisition of all the confirmations from all the other Proposer agents. However, this approach is guaranteed to find a useful solution, i.e., the best one for the Proposer, if it exists. The host of each meeting will check all possible dates from the most preferred to the less preferred one to schedule its meeting. Nevertheless, to prove the termination of this approach we have to prove that the underlying protocol never goes into an infinite loop while scheduling a meeting.

Let's assume that this approach goes into an infinite loop while scheduling a meeting. To schedule a meeting X_l^i all the participants will cooperate together to find the best date for this meeting. The system will go into an infinite loop while scheduling X_l^i if and only if the Participant agents reprocess the checked dates (cycle) when no solution is found. However, the number of possible dates meeting is discrete and finite. Moreover, every unsuccessfully checked date is removed from the system to avoid returning to it later. The system will stop when a "good" date is found or when all possible dates for X_l^i are processed and no possible solution has been found. Hence our assumption is not true.

We have to note that the satisfaction state of all the agents in a distributed system can be achieved by taking snapshots of the system, using the well-known algorithm of Chandy-Lamport in [19]. Termination occurs when all agents are waiting for a message and there is no message in the transmission channels. The cost of the termination process can be mitigated by combining snapshot messages with our protocol messages.

7.4.2 Spatial and temporal complexity

Let us consider an MS problem implying n for total number of users, d for the maximal number of possible dates per meeting, $|C_H| = c_H$ for the maximum number of preferred dates per user and $|C_S| = c_S$ for the maximum number of non-available time slots per user. The total number of agents in this system is n the same as the total number of users. Suppose that each meeting involves n attendees and each user has m already scheduled meetings in his/her calendar. Let's compute the complexity of adding a new meeting into the existing schedule.

The solving process of the proposed scenario is divided into two steps. In the first step, the pruning step, the initiator agent will perform $O((c_H+m)d)$ operations to filter the slot times of the new event. Then, this agent will transmit the obtained set of possible remaining dates to the (n-1) attendees to carry out the same process. The time complexity of this step, in the worst case, is $O(nd(c_H+m))$.

For the second step, the initiator agent will first determine the intersection of the sets of the received times, leading to $(n\text{-}1)d^2$ operations, then choose one proposal among d dates , according to the proposed optimal criteria; this process requires $O(c_S d\log(d))$ operations. The agent will send its proposal to the attendees to check it. Each attendee will perform $O(m)$ operations to check if it has received in the meanwhile another proposal for the same date.

Then the total temporal complexity of the second step is O((*n*-1)d^2+$c_S d\log(d)$+ *nm*). Finally, the temporal complexity for each new coming event is, O($nd(c_H+m)+nd^2+ c_S d\log(d)+nm$) in the worst case. The spatial complexity for all the agents is O($n(c_S+c_H+d+m)$) in the worst case, where (c_S+c_H+*d*+*m*) is the total size of the initial calendar for each agent.

7.5 Experimental comparative evaluations

To evaluate the proposed approach, we have developed the multi-agent dynamic with Actalk, an object oriented concurrent programming language using the Smalltalk-80 environment. In our experiment, we generated random meeting scheduling problems. The parameters used for a meeting problem are: *n* agents in the system, *m* meetings per agent, *p* participants in a meeting, *D* global calendar, c_H number of initial hard constraints per agent, c_S number of initial soft constraints per agent, *d* maximal possible time slots per event, $w_k^{A_i}$ weights for the soft constraints, and $W_{X_k}^{A_i}$ weights of the meetings (the weight of each hard constraint is equal to 1).

In order to compare our approach with that reported by Tsuruta and Shintani in [101], we used the same parameters to run both algorithms on randomly generated samples. Note that the approach in [101] presents some restrictions towards; first the handle of the hard constraints (i.e., all the constraints could be relaxed by this approach) and second, the discrimination between meetings. This approach processes all the proposed meetings with the same importance independently of neither the proposer nor the attendants. However in the real world, all meetings are not equivalent. For this reason we have brought to our consideration the notion of meeting priority in our formalization by associating a weight $W_{X_k}^{A_i}$ to reflect its greatness. Our approach tries then, in its solving process, to *first* schedule the most important meeting maintained by each agent, unlike the approach in [101].

In this manner, we attempt to describe ideally the real world meeting scheduling problems. Therefore two kinds of experimentation are given in this section. For the first kind, we assume that for each generated problem, we have only soft constraints. We carried out the two approaches on the same meeting instances with: *n*=15, $m \in \{3, 5\}$, *p* = 10, $c_S \in \{20, 40, 60, 80, 100\}$, and $W_{X_k}^{A_i} \in [0..1]$, $w_k^{A_i} \in [0..1]$ were randomly chosen. (35 instances are generated for each $\langle m; c_S \rangle$). The initial calendar *D* in each problem is equal to 100.

Table 7.1 shows the obtained mean results for the ratio of CPU time of the approach in [101] divided by the CPU time of MSS approach. In order to analyze these results, let us consider the case $\langle 3; 20 \rangle$. Both approaches require almost the same CPU time. For this case, the number of possible time slots is not large leading to a few number of constraints, so the approach in [101] can rapidly find a solution for each meeting without relaxing many constraints, causing few iterations on the same meeting. However, when the amount of soft constraints increases, the needed time for the approach in [101] almost double. With regard to MSS, the increment in the CPU time is largely lower than Tsuruta et al. approach. Obviously, the main explanation of this result is that For Tsuruta et al. approach (see chapter

4), the agents of the system (the group agent and the participant agents) should exchange the possible constraints according to the time slots of the current meeting to schedule. As the number of meeting constraints grows, and so does the probability of getting the same dates for the meetings. Therefore, the number of relaxed constraints by the approach in [101] increases leading to additional iterations for the same meeting and hence an increase in the CPU time.

Table 7.1. Mean results of MSS approach and Tsuruta et al. approach in term of the CPU time for meeting problems without hard constraints. *Ratio CPU* **= CPU time Tsuruta et al. approach / CPU time MSS.**

	$\langle$**3; 20**$\rangle$	$\langle$**3; 40**$\rangle$	$\langle$**3; 60**$\rangle$	$\langle$**3; 80**$\rangle$	$\langle$**3; 100**$\rangle$
CPU time Tsuruta et al. App.	604.57	1894.74	4479.89	8378.06	13781.29
CPU time MSS	452.49	520.43	617.63	724.40	863.09
Ratio CPU	1.34	3.64	7.25	11.57	15.97
	$\langle$**5; 20**$\rangle$	$\langle$**5; 40**$\rangle$	$\langle$**5; 60**$\rangle$	$\langle$**5; 80**$\rangle$	$\langle$**5; 100**$\rangle$
CPU time Tsuruta et al. App.	2116.14	6471.34	14849.86	23997.46	38355.91
CPU time MSS	776.14	871.71	1020.00	1204.80	1466.51
Ratio CPU	2.73	7.42	14.56	19.92	26.15

In addition, The protocol proposed by Tsuruta et al. proceeds by setting up a threshold equal to zero for the constraints to relax and then through negotiations with personnel agents, the group agent tries to relax the constraints, i.e., by incrementing the threshold, until attaining a compromise among participants. while our approach relies on a *selfish* protocol. We try to find the solution that maximizes the Proposer's preferences. Although in our approach; we try to process the most important meetings at first.

The approach in [101] takes more time than our approach in most cases because both the number of constraints and the number of meetings grow. Furthermore, in MSS approach each agent tries to perform all its meetings asynchronously and in parallel while for the approach in [101], it is done in a synchronous sequential manner for the same group agent[4].

As for the second kind of experimentation, i.e., to appraise the greatness of the enforcement of local consistency in the solving meeting problems, we have chosen to measure the percentage of reduction made by the first step of our approach. For this purpose, examples including hard constraints were randomly generated with n=10, m=3, $p \in \{3, 5, 7\}$, $c_H \in \{0,$ 10, 20, 30, 40, 50$\}$ and $d \in \{100\%, 83\%, 66\%, 50\%, 33\%, 16\%\}$ corresponding respectively to each c_H. For each pair $\langle p, c_H \rangle$, we first generated 35 instances, then we measured the average of the achieved results.

[4]Meetings are proceeded in parallel within group agents.

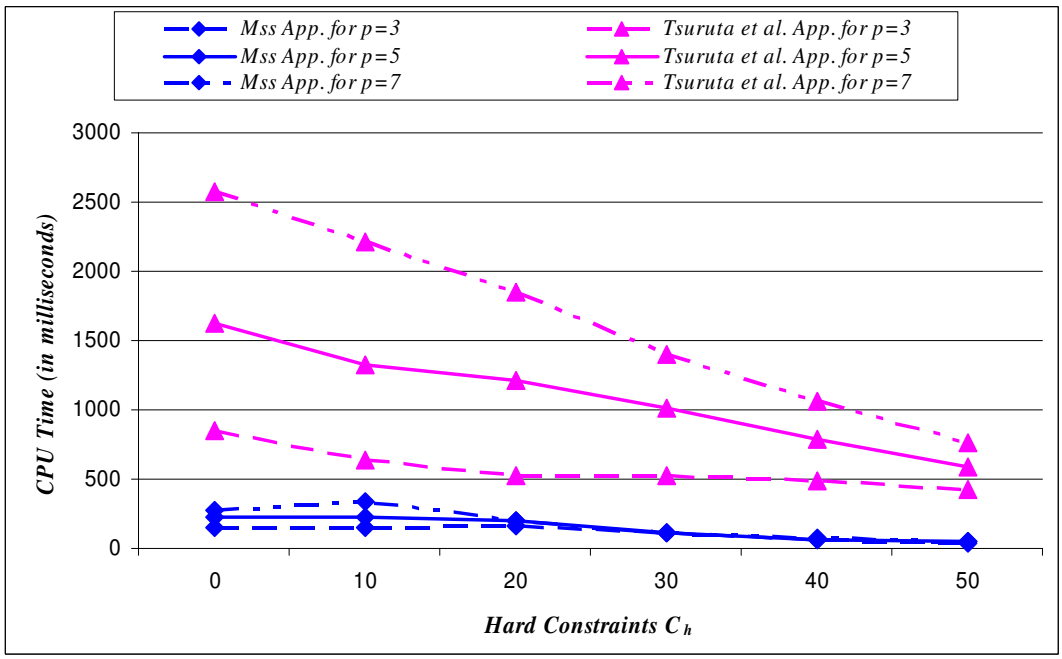


Figure 7.2. MSS approach vs. Tsuruta et al. approach in term of mean of the required CPU time in milliseconds. (35 random samples generated for each pair $\langle C_h, p \rangle$).

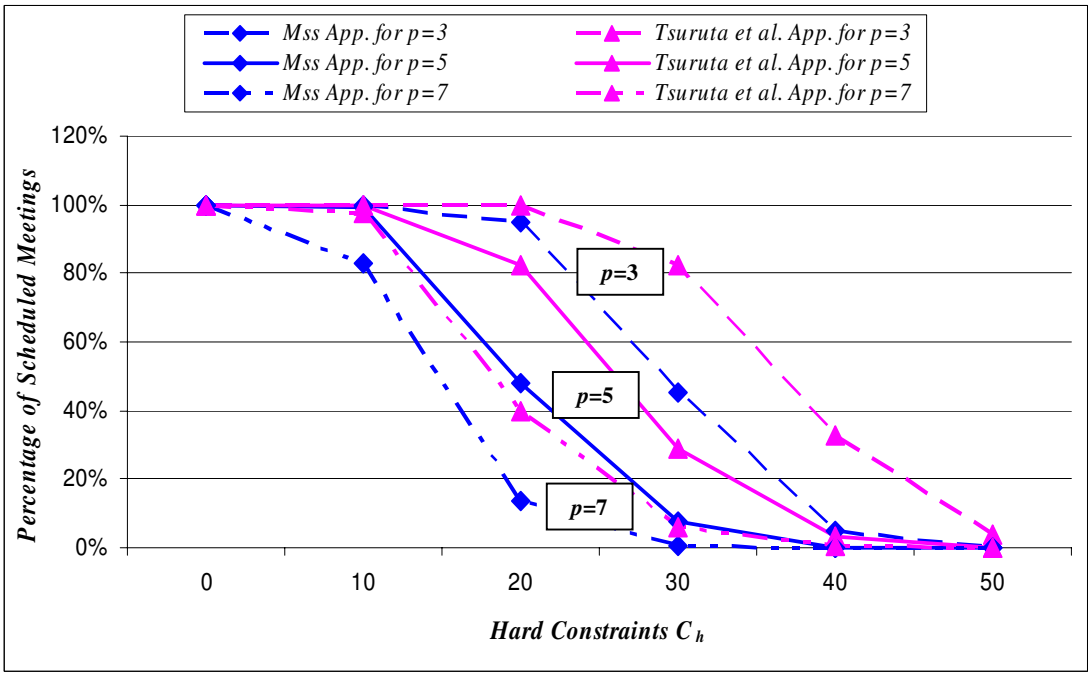


Figure 7.3. MSS approach vs. Tsuruta et al. approach mean results in terms of the percentage of scheduled meetings.(35 random samples generated for each pair $\langle C_h, p \rangle$).

These results are expressed in term of five criteria: (*i*) the CPU time spent by each of the two approaches, (*ii*) the percentage of scheduled meetings, (*iii*) the percentage of reduced soft constraints performed by the first step of the proposed approach, (*iv*) the required number of messages passed and (*v*) the amount of exchanged information. We have introduced some modifications to the approach in [101] to make it worthwhile for both hard and soft constraints. These two approaches were carried out on the same meeting examples. Figures 7.2 and 7.3 show the achieved mean results of both approaches in term of CPU time and the percentage of scheduled meetings. These results show that our approach requires less CPU time than approach [101]. For example in the case of 7 participants and 50 hard constraints, the problem is over-constrained and thus no meetings can be planned, i.e., no agreement can be reached between all the attendants. Therefore, our approach can discover merely the absence of solution from the first step, and before starting the solving process.

In the case of 7 participants and 20 hard constraints, only a few meetings can be planned. Table 7.2 shows that the percentage of pruned dates from possible ones is high (=96,66%). Therefore, our approach is able to schedule the possible meetings in considerably less CPU time than approach [101], i.e., the approach in [101] requires more than five times the time needed by our approach. This result can be elucidated by the fact that the first step is useful in order to discard the dates that cannot be in any solution and consequently avoid exploiting them in the solving process, leading to decreased CPU time consumption.

Table 7.2. MSS approach mean results in term of the percentage of reduced time slots for each pair $\langle p, c_H \rangle$.

	$\langle$**3; 0**$\rangle$	$\langle$**3; 10**$\rangle$	$\langle$**3; 20**$\rangle$	$\langle$**3; 30**$\rangle$	$\langle$**3; 40**$\rangle$	$\langle$**3; 50**$\rangle$
% ***Reduction***	0.00%	51.44%	80.17%	94.04%	99.13%	99.98%
	$\langle$**5; 0**$\rangle$	$\langle$**5; 10**$\rangle$	$\langle$**5; 20**$\rangle$	$\langle$**5; 30**$\rangle$	$\langle$**5; 40**$\rangle$	$\langle$**5; 50**$\rangle$
% ***Reduction***	0.00%	67.25%	91.49%	98.82%	99.95%	100.00%
	$\langle$**7; 0**$\rangle$	$\langle$**7; 10**$\rangle$	$\langle$**7; 20**$\rangle$	$\langle$**7; 30**$\rangle$	$\langle$**7; 40**$\rangle$	$\langle$**7; 50**$\rangle$
% ***Reduction***	0.00%	77.14%	96.66%	99.72%	100.00%	100.00%

Nevertheless, for the percentage of the meetings scheduled, the approach in [101] planned, for some cases, more meetings than our approach. This is defended by the fact that for our approach we tried to plan the most important meeting *at first*. For example, in the case $\langle 7; 20 \rangle$ the 40% of the meetings scheduled by the approach in [101] may or may not contain the most important meetings in the problem. Meanwhile, with our approach we are sure that the 14% of the meetings scheduled are the most important because they were first chosen to be processed using their weights.

As for the number of exchanged messages needed to reach an agreement among all the

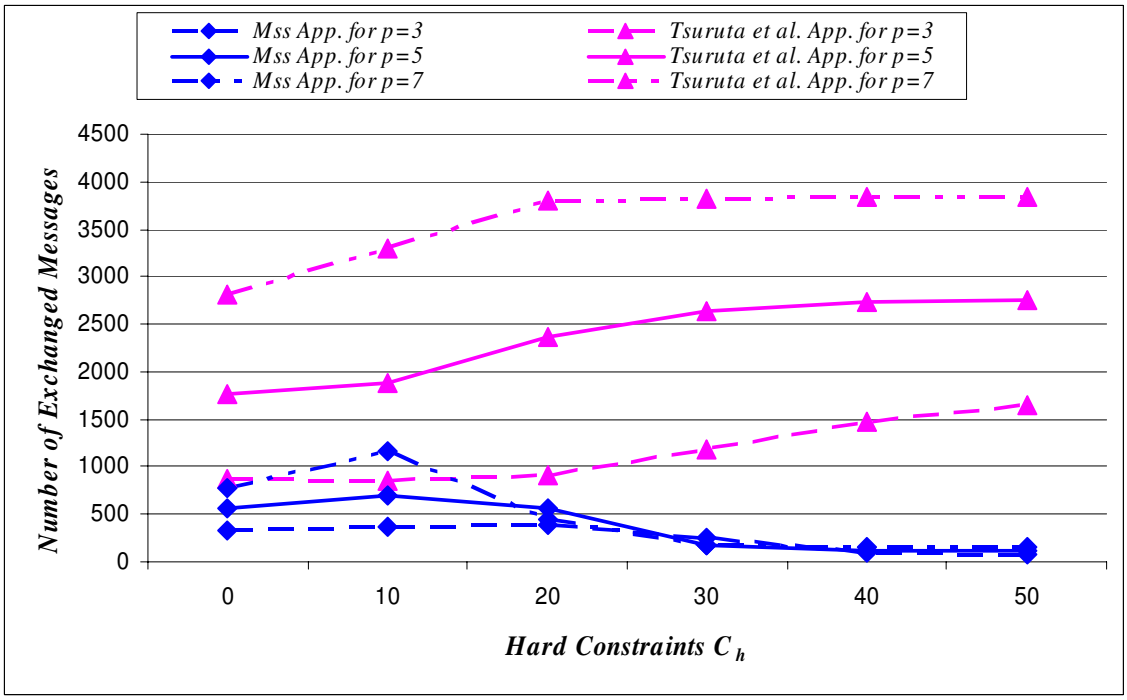


Figure 7.4. Mean results in term of the number of exchanged messages.

users, Figure7.4 shows that the proposed approach requires many fewer exchanged messages than the Tsuruta approach [101]. This number increases with the number of participants in the meeting, even if the problem has no solution, i.e., there is no possible time at which all the participants can be gathered. However, with the proposed approach, the percentage of reduced values increases with both the number of participants and the number of hard constraints, consequently the number of exchanged messages decreases.

Finally, concerning the size of exchanged messages, we measured the required amount of information to reach a consensus among participants for both approaches. Figure 7.5 shows that in all cases MSS transfer less information than Tsuruta et al. approach. Let's recall that for Tsuruta et al. approach, the negotiation process is based essentially on sending constraints related the user. The amount of the transferred information decreases at each step of the iteration process. Nevertheless, this amount largely increases with the number of participants in the meeting. Regarding MSS approach, only during first step agents need to exchange all the possible time slots for the concerned meeting. This step is necessary in order to reduce the set of meeting's dates and consequently to avoid as much as possible the checking of initially non-valid dates.

We performed statistical hypothesis testing to evaluate the fairness of the obtained random experimental results. In order to compare the means of the samples' results obtained for both approaches in terms of CPU time and percentage of scheduled meetings, we measured them using Matlab6.1 the dependent two samples t-test with significant level α = 0.05.
For the mean in CPU time, we formalized the null hypothesis H_0 and the alternative hypoth-

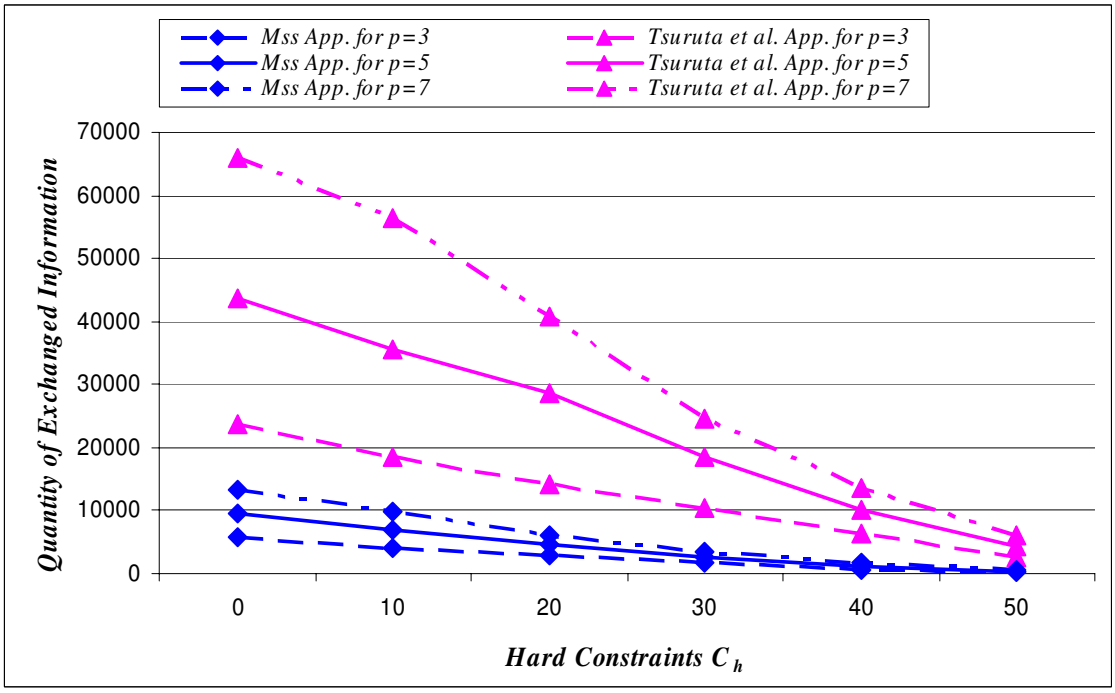


Figure 7.5. Mean results in term of the necessary amount of exchanged information, i.e., necessary number of slot times exchanged to reach an agreement.

esis H_1 as follows:

H_0: $\mu_{MSS\ App.} = \mu_{Tsuruta\ et\ al.\ App.}$

H_1: $\mu_{MSS\ App.} < \mu_{Tsuruta\ et\ al.\ App.}$

In all cases, i.e., for each pair $\langle c_H, p \rangle$, with 34 degree of freedom, the null hypothesis (H_0)) is rejected with significance equal to zero, which means that it never happens even *by chance* that the observed value of T-statistic could be as larger or larger with confidence interval 95%.

As for the means in term of the percentage of scheduled meetings, we carried on the following hypothesis testing:

H_0: $\mu_{MSS\ App.} = \mu_{Tsuruta\ et\ al.\ App.}$

H_1: $\mu_{MSS\ App.} \neq \mu_{Tsuruta\ et\ al.\ App.}$

For the cases $\langle 0, 3 \rangle$, $\langle 0, 10 \rangle$, $\langle 0, 5 \rangle$, $\langle 50, 0 \rangle$, $\langle 0, 7 \rangle$ and $\langle 50, 7 \rangle$ the percentages of scheduled meetings are exactly the same for both approaches. However for the remaining cases and according to the obtained two samples t-test result, the null hypothesis is rejected for all pair $\langle c_H, p \rangle$ with maximum significance=0.0399 for $\langle 40, 7 \rangle$ which means that for this case by chance we would have observed values of T more extreme that the one in this example in

only 399 of 10000 similar experiments. A 95 % confidence interval on the mean is [-0.2232 -0.0054], which includes the theoretical (and hypothesized) difference of both means. Also, same for the case $\langle 10, 5 \rangle$, the significance is higher than other cases equal to 0.0037. Where as all the other cases, the significance is very low or equal to 0.

We can conclude that our approach is a scalable approach that outperforms the approach described in [101] and this is especially true when the number of meetings and the number of participants increase. One must note also that our approach seems to be more appropriate to real-world applications by dealing especially with strong constraints (i.e., inequality) and by bringing forward consideration of discrimination among the proposed meetings. In addition, in our approach, agents do not reveal directly any of the information related to the human user even to a trust person. While the approach in [101] relies especially on exchanging users' private information with the group agent, no level of privacy is preserved.

The first step of the MSS approach can fulfill a premature detection of the impossibility of reaching any agreement between all the participants and this by maintaining arc-consistency.

7.6 Summary

In this chapter we proposed a new agent-based solver for any meeting scheduling (MS) problems with predictable structure that reflects ideally real-world applications. To fulfill such condition, we have considered, in our model, two kinds of constraints to model the users' requirements: hard constraints to model the non-availability of a user and soft constraints to define his/her preferences. The underlying multi-agent architecture associates a User agent to each user and makes them interact by sending asynchronous point-to-point messages containing only relevant information to keep, as much as possible, their privacy. The basic idea of this approach consists of two steps: the first reduces the initial problem by reinforcing some levels of local consistency (node and arc consistency); and the second step solves the resulting meeting scheduling problem with a minimum amount of exchanged messages.

This approach was implemented with Actalk under the Smalltalk-80 environment and compared with an existing approach in literature described in [101] on randomly generated instances, in mean of CPU time, percentage of scheduled meetings, the number of exchanged messages and the amount of transferred information. The obtained results show that our approach is scalable and worthwhile for processing strong constraints. In addition, in order to show the importance of the first step, i.e., reduction step, other experiments were made to measure the percentage of non-viable values discarded from the meetings' calendars. The obtained results showed that this process is appropriate for reducing the static MS problem and consequently the search space, without loss of solutions. This work has been published in [2, 7, 9].

Chapter 8

MSRAC: Dynamic Meeting Scheduling Solver

In the previous chapter, we described a novel approach to solve any static MS problem. However, for some organization, knowing all the meetings in advance might be quiet difficult rather impossible. Therefore, we focused our new research on solving any MS that are subject to many alterations, i.e., adding of new meeting and/or cancelation of an already scheduled one. In addition, we focus our second objective on minimizing the amount of exchanged messages by virtue of the real difficulty of messages passing operations in distributed systems.

In this chapter we present a novel, scalable, dynamic and entirely distributed solution for MS problems that accounts for user preferences, handles several events with various levels of importance and especially minimizes the number of exchanged messages. In the sequel we introduce first the new formalization of any *dynamic* MS problem. Second we describe the proposed agent-based model. Then, we introduce the global dynamic followed by an illustration of the new protocol through an example. Finally we give a theoretical and empirical evaluation of the new approach followed by a summary of this chapter.

8.1 Dynamic meeting scheduling problem formalization

We formalize the dynamic MS problem as a DVCSP (dynamic valued constraint satisfaction problem). Like the previous static approach in Chapter7, each user maintains two kinds of constraints: hard and soft constraints related to him/her, along with other strong constraints defining the specific features of the problem itself. Let's recall these two constraints in the following.

Definition 39 *We define a dynamic MS problem, as a DVCSP, by a sequence of quintuples (X, D, C,* S, φ*) where*

- *X =* $\{X_1, \ldots, X_n\}$ *is the set of* n *meetings that need to be scheduled at an instant* t*.* X_k *with* $k \in \{1, \ldots\ n\}$ *denotes the* k^{th} *meeting to schedule.*

- *$D = \{D_1, \ldots, D_n\}$ is the set of all possible dates for all the meetings X. D_i=$\{dt_{i_1}, \ldots, dt_{i_d}\}$ (with $|D_i|$=d), is the set of possible dates for the meeting X_i.*

- *C is the set of all constraints of the problem. C is composed of the following constraints:*

  - hard constraints*: represented by, on the one hand, C_h the set of the constraints related to the non-availability of all users (see the white box in Figure 8.1). On the other hand, $C_{allDiff}$ the set of allDiff constraints relating each pair of meetings X_k and X_l sharing at least one participant A_j ($A_j \in$ Part(X_k) and $A_j \in$ Part(X_l)[1].).*
  - soft constraints, *C_s is the set of the soft constraints related to the preferences of all users towards the possible dates in their calendar (see the gray box in Figure 8.1).*

- *$\varphi : C \rightarrow E$. C=$\{C_h \cap C_{allDiff} \cap C_s\}$, for each hard constraint $c_i \in \{C_h \cap C_{allDiff}\}$ we associate a weight $\perp$ and for each soft constraint $c_j \in C_s$ we associate a weight* $w_j \in$ *[0..1][2]. This weight reports the degree of preference of a user to have a meeting at the date* dt_j *(see the number inside the gray box in Figure 8.1).*

- S *represents the valuation structure that defines the proposed optimality criteria (discussed in next section) and will be used to find the "best" solution.*

In addition, for each meeting X_k we assign a different weight $W_{X_k} \in$ *[0..1] to define the degree of importance of X_k ($k \in \{1, \ldots, n\}$) and it is used to allow the processing of the* most important *meeting*[3] *at first.*

Solving a MS problem consists in finding a "good" assignment *sl** $\in$ *Sol*:=$D_1 \times \ldots \times D_n$ of the variables in X=$\{X_1, \ldots, X_n\}$ according to their importance W_{X_k}, such that all the hard constraints are satisfied while maximizing the global utility (*GU*) of all the users for all the scheduled meetings such that:

$$sl^* = arg\ max_{sl \in Sol} GU(sl) \tag{8.1}$$

The computation of the *GU* will be given in detail in the next section.

[1]The function *Part*(X_k) denotes all the participants in the meeting X_k

[2]This assumption does not contradict the ability of our protocol to support any kind of preferences' measurement evaluation.

[3]We assume that the users report truly and accurately the importance of their meetings.

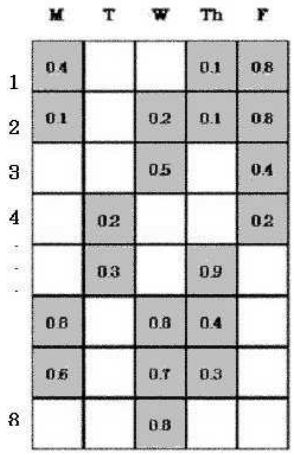


Figure 8.1. Example of a user calendar.

8.2 MSRAC multi-agent model

The proposed MSRAC model, is the same as proposed previously for static MS problem in Chapter7, involving User agents and an Interface agent. Each User agent has its own acquaintances, own knowledge (static and dynamic) and a reasoning engine. The acquaintances of a User agent A_i are dynamic and depend on the current meeting to be scheduled (or rescheduled). At an instant *t*, the acquaintances of A_i are defined by all the participants User agents in the current meeting X_k. The static knowledge of a User agent is formed by the possible dates for the underlying meeting X_k and the user's constraints. Its dynamic knowledge is formed by both its acquaintances and its current calendar.

All the User agents will negotiate and cooperate together to schedule all the meetings proposed by the human users. Therefore we assume as for the static approach the following communication model between all agents:

- The agents in the system negotiate by exchanging asynchronous point-to-point messages containing the necessary relevant information in a manner that reduces the number of messages passing and keeps the most privacy for the involved users.
- An agent can send a message to another only if it knows that this agent belongs to its acquaintances.
- The messages are received in a finite delivery time and in the same order that they are sent. Messages sent from different agents to a single agent may be received in any order.

For efficiency, the proposed approach tolerates parallel execution, i.e., more than one meeting can be processed at the same time.

Each User agent A_i ($A_i \in$ A) maintains a sequence of $\text{VCSP}_P^{A_i}$ (X^{A_i}, D^{A_i}, C^{A_i}, *S*, φ) for which the set of variables $X^{A_i} \in X$ represents the user's A_i meetings to schedule at the instant

t. The constraints $C^{A_i} \in C$ ($C^{A_i} = \{C_h^{A_i} \cup C_s^{A_i} \cup C_{allDiff}^{A_i}\}$) represent the non-availability, the calendar of this user and the constraints relating each pair of meetings.

In this multi-agent model, the intra-agent constraints are defined by the aforementioned constraints, whilst the inter-agent constraints are represented by the set of strong constraints, i.e. *equality* constraints. An equality constraint exists between two User agents A_i and A_j if and only if there exist at least one meeting $X_k^{A_i}$ (resp. $X_l^{A_j}$) such that $A_j \in$ *Part*($X_k^{A_i}$) (resp. $A_i \in$ *Part*($X_l^{A_j}$)). We should discern that the equality constraints are dynamic.

The local goal of each User agent A_i is to schedule all its meetings, whenever possible, such that on the one hand all its hard constraints $C_h^{A_i} \cup C_{allDiff}^{A_i}$ are satisfied, and on the other hand the higher local utility (*LU*) for all the planned meetings is achieved. The *LU* brought off by a meeting $X_k^{A_i}$ scheduled at the date $dt_p \in D_k^{A_i}$ ($LU(X_k^{A_i}, dt_p)$ is defined by the summation of the preferences (soft constraints) of all the participants $A_j \in$ *Part*($X_k^{A_i}$) (Eq.8.2)

$$LU(X_k^{A_i}, dt_p) = \sum_{A_j \in Part(X_k^{A_i})} w_p^{A_j} \tag{8.2}$$

In order to fulfill its local goal, each User agent A_i should choose for each of its meetings $X_k^{A_i} \in X^{A_i}$ the date dt_p that maximizes its *LU* (Eq.8.3)

$$\begin{array}{ll} \max & LU(X_k^{A_i}, dt_p) \\ dt_p \in D_k^{A_i}; & \\ p \in \{1, \ldots, |D_k^{A_i}|\} & \end{array} \tag{8.3}$$

The global goal of the whole system is to schedule the maximum of the meetings of all the User agents satisfying all the inter-agent constraints and achieving the higher global utility (*GU*) which defines the quality of the solution. The *GU* is represented by the summation of all local utilities corresponding to the planned meetings (in the set of possible solutions *s*) by using Eq.8.4.

$$GU(sl) = \sum_{A_i \in A} \sum_{\substack{(X_l^{A_j}, dt_p) \in sl; \\ dt_p \in D_l^{A_j}}} LU(X_l^{A_j}, dt_p) \tag{8.4}$$

However, for any meeting $X_k^{A_i}$, a date $dt_p \in D_k^{A_i}$ may be the most preferred by one participant and non-preferred (or less preferred) by the other participants. Therefore, in order to guarantee the maximum preference similarities between all the participants we propose to add to our system another criterion to satisfy this condition. The idea is to choose the date that, in addition to the first criterion (Eq.8.4), minimizes the distance between the own users'

preferences by using Eq.8.5.

$$\min_{\substack{dt_p \in D_l^{A_j}; \\ p \in \{1,\ldots,|D_l^{A_j}|\}}} \max_{A_k, A_i \in Part(X_l^{A_j})} |w_p^{A_k} - w_p^{A_i}| \tag{8.5}$$

To illustrate the use of Eq.8.5 more clearly, let us consider the following example of 4 User agents (users) given the task of scheduling one meeting. We assume that A_1 is the Proposer of this meeting $X_1^{A_1}$ and all the other User agents are the participants. The possible dates for $X_1^{A_1}$ are $D_1^{A_1}$={(Tu, 7), (Wed, 2), (Wed, 7), (Th, 2), and (Th, 6)}. Table **??** illustrates the preferences $w_1^{A_i}$ of each attendee $A_i \in Part(X_1^{A_1})$ toward each date $dt_p \in D_1^{A_1}$.

Table 8.1. Example of the degree of preference of each user A_i towards each possible date dt_p for the meeting $X_1^{A_1}$.

	(Tu, 7)	**(Wed, 2)**	**(Wed, 7)**	**(Th, 2)**	**(Th, 6)**
$\boldsymbol{A_1}$	0.1	0.3	0.9	0.6	0.4
$\boldsymbol{A_2}$	0.6	0.7	0.3	0.5	0.4
$\boldsymbol{A_3}$	0.1	0.2	0.3	0.6	0.7
$\boldsymbol{A_4}$	0.7	0.3	0.4	0.2	0.1
$\boldsymbol{LU(X_1^{A_1}, dt_p)}$	1.5	1.5	**1.9**	**1.9**	1.6

However, according to Table 8.1, the dates (Wed, 7) and (Th, 2) maximize the utility (LU) of the meeting, i.e., the sum of the utilities of all attendees for both of the two dates is 1.9. If we adopt the same strategy as [41], the optimal solution should be the date (Wed, 7), with 0.3 as the overall preference. But this date is the most preferred only by A_1, while it is the less preferred by A_2, A_3, A_4. Thus with the second criterion we should instead chose the date (Th, 2), because it minimizes the difference between the users' preferences (max{|0.9-0.3|; |0.6-0.2|}), and consequently reinforces the similarity between the attendees[4].

It is noteworthy that the above optimality criteria is based essentially on the preferences of the attendee toward the possible dates of the underlying meeting. Such criteria require a common preferences scale otherwise it is not fair to compare the personal preferences of the participants in a meeting. To satisfy this condition without forcing the participants to reveal their private Calendar, we propose to integrate a new heuristic in the solving process. This heuristic allows the use of any ordering or scale to express the preferences of users (no common scale is imposed on users to express their own preferences). It is worth remarking at this stage that the use of such optimization criteria may lead to the classical problem of

[4]We suppose that all the attendees have the same level in the company.

constructing interpersonal utilities functions [40], i.e., *how to compare users' preferences using independent and different ordering and/or measurement scales ?*.

In this paper, the used criteria do not require any common ordering or scale over all the agents to express their preferences. The basic idea is to ask each attendee A_j in a meeting $X_k^{A_i}$ to *rank* the set of possible dates for $X_k^{A_i}$ from the most to the less preferred, i.e., $dt_p \prec dt_l$ if and only if $w_p^{A_i} > w_l^{A_i}$. For the previous example, the User agent A_2 will rank the possible dates for $X_1^{A_1}$ as follows: (Wed, 7)≺(Th, 2)≺(Th, 6)≺(Wed, 2)≺(tu, 7). Then the Proposer agent will generate a new implicit ordinal scale[5] as stated by the received ordered sets. The lowest date dt_p in the order has the greatest number of votes associated with it. The Proposer agent will fist assign an implicit preferences $Iw_p^{A_i}$ to each dt_p and then use it to determine the best date. Table 8.2 presents the candidate dates and their implicit preferences generated by the Proposer agent.

Table 8.2. Example of users' implicit preferences generated by the Proposer agent.

	A_1	A_2	A_3	A_4
5	(Wed, 7)	(Wed, 2)	(Th, 6)	(Tu, 7)
4	**(Th, 2)**	(Tu, 7)	**(Th, 2)**	(Wed, 7)
3	(Th, 6)	**(Th, 2)**	(Wed, 7)	(Wed, 2)
2	(Wed, 2)	(Th, 6)	(Wed, 2)	**(Th, 2)**
1	(Tu, 7)	(Wed, 7)	(Tu, 7)	(Th, 6)

In this example the local utilities[6] of the two candidates (Wed, 7) and (Th, 2) are the same (Table 8.3). The Proposer agent will choose (Th, 2) to enforce the similarity[7] between the participants. The maximum difference for (Wed, 7) is |5-1|=4, while it is |4-2| for (Th, 2).

Table 8.3. Example of LU computation for each candidate.

Candidate dates	**Local Utility**
(Tu, 7)	1+4+1+5=11
(Wed, 2)	2+5+2+3=12
(Wed, 7)	5+1+3+4=13
(Th, 2)	4+3+4+2=13
(Th, 6)	3+2+5+1=11

It is noteworthy that this pseudo-common scale is dynamic, and may change according to the candidate dates for a meeting. Hence, the local utility of a meeting should be normalized to compare it to another one with different scale.

[5]This idea cannot handle cardinal preferences

[6]Computed according to Eq.8.2

[7]According to Eq.8.5

When there is a conflict between two meetings $X_k^{A_i}$ and $X_l^{A_j}$ for two different User agents or for the same agents ($X_k^{A_i}$ and $X_l^{A_i}$) ($W_{X_k}^{A_i} = W_{X_l}^{A_j}$), three issues (deterministic and non-deterministic) can be applied to solve the conflict. If User agent A_i, which has a meeting already scheduled $X_k^{A_i}$ in its calendar at the date d_p, receives another meeting $X_l^{A_j}$ with the same importance to be scheduled at the same date d_p, then it will choose on of the following issues:

1. The deterministic issue, defined as always scheduling at d_p the meeting that will increase LU, i.e., If ($LU(X_k^{A_i}, dt_p) > LU(X_l^{A_j}, dt_p)$) then $X_k^{A_i}$ will be scheduled at dt_p and $X_l^{A_j}$ will be scheduled at another date $dt_h \neq dt_p$. Otherwise inversely.

2. The non-deterministic issue, consists in arbitrarily choosing one of the meetings in conflict ($X_k^{A_i}$ or $X_l^{A_j}$) to schedule at d_p and rescheduling the other one.

3. The second non-deterministic issue, defined as using the *metropolis criterion* in order to choose the meeting to reschedule. $X_h^{A_i}$ is accepted to be scheduled at date dt_p by applying the following acceptance probability (Eq.8.6). Notice that this process leads to the rescheduling of $X_l^{A_j}$ and perhaps to the rescheduling of other meetings (with less importance) by propagation. The main idea behind using metropolis criterion to solve the conflict is that trying always to increase *LU* may not lead to the optimal solution, while accepting some deterioration in the *LU* may increase the final *GU*.

$$P_c\{\text{ accept } X_k^{A_i} = dt_p\} = \begin{cases} 1 \text{ if } LU(X_k^{A_i}, dt_p) \geq LU(X_l^{A_j}, dt_p) \\ Exp(\frac{LU(X_k^{A_i}, dt_p) - LU(X_l^{A_j}, dt_p)}{Tp}) \text{ otherwise} \end{cases} \quad (8.6)$$

   where $Tp \in R^+$ denotes the temperature.

In the sequel, the User agent that proposes the meeting is called the Proposer agent. The same User agent can be both a Proposer agent and a Participant agent at the same time.

It is noteworthy that our focus in this work, as mentioned in [42], was to find a good compromise between three main features: minimizing privacy loss, maximizing solution quality, and minimizing the required time to achieve it. Therefore we propose to integrate, in our protocol, the above mentioned heuristic to choose the best solution, according to the aforementioned optimality criteria, without revealing the *real* preference values of the corresponding participants.

8.3 MSRAC global dynamic

The global objective of our proposed approach is to schedule all meetings for all users, while maximizing the global utility and ensuring near fulfillment of users' preferences. The multi-agent meeting scheduling negotiation protocol is divided into two steps.

- Step 1. Use the basic idea of the DRAC approach to transform the initial MS problem into another equivalent MS' by enforcing local consistency [65] (node and arc consistency). MS' is obtained as a result of the interactions between the Proposer agent and the Participant agents. The primary objective for this step is to benefit from the main goal of DRAC in order to make the search for a global solution of any MS problem easier while saving futile backtracking, i.e., to avoid changing the previous chosen meeting's date.

- Step 2. To solve the obtained MS problem via interactions and negotiations between agents. Each agent that has a meeting to schedule (Proposer agent) searches for the best solution for its meeting that, on the one hand, satisfies the two conditions given in Section 3, and on the other hand, satisfies all of the Participant agents' constraints. More than one meeting can be processed in parallel (by different Proposer agents). Thus, the same agent can be, at the same time, a Proposer agent (to fix its meeting) and a participant agent (in different meeting proposed by another Proposer). This system does not include any central node to process meetings.

Before introducing the global dynamic, we present the communication protocol.

8.3.1 Communication protocol

For the communication protocol, the two basic message-passing primitives used for each agent are the same as those used in the DRAC approach (see Section 3).

- *sendMsg*(Sender, Receiver, *Message*) is used to send a message to one or more receivers.

- *getMsg*() extracts the first message from the mailbox of the agent.

With respect to exchanging messages, the underlying Multi-Agent dynamic involves the following messages:

- *"Start"* message, sent by the Interface agent to the corresponding Proposer agent to activate it whenever there is a new meeting given by a user.

- *"RedMeetCalendar: with:"* message, sent by a Proposer agent to each Participant agent to ask it to adjust the possible dates of the meeting according to both its user's non-availability and its calendar.

- *"Reply"* message, sent by each Participant agent to the Proposer agent in order to propagate its reductions.

- *"ReceiveProp: with:"* message, sent by the Proposer agent to the Participant agent to verify the viability of the proposal.

- *"MeetNotPossible"* message, sent by each agent to the sender agent to inform it about the non-possibility of the meeting.
- *"MeetingOK"* message, sent by each agent to the sender agent to inform it about its agreement for the date of the meeting.
- *"UpdateProp: with:"* message, sent by a Participant agent to a Proposer agent, in the case of conflict between two (or more) meetings, in order to invite it to relax its preferences.

8.3.2 Multi-agent interaction protocol for dynamic MS

A user who wants to host a meeting must tailor the Interface agent, which will activate the corresponding Proposer agent and make it interact with all of the Participant agents. Note that more than one Proposer agent can be activated at the same time. Each activated Proposer agent A_i must first reduce the set of possible dates of the corresponding meeting $X_k^{A_i}$ according to its hard constraints. This process can be viewed as node consistency reinforcement and aims to reduce the possible candidates for a meeting by eliminating those dates on which this meeting cannot be held. If the set of possible dates of the meeting becomes empty after reduction then its possible dates must be changed. Otherwise, the Proposer agent must delete all the dates that were used for more important meetings, i.e., all meetings ($X_l^{A_j}$, d_p) $\in$ *Calendar*^{A_i} for which $W_{X_k} < W_{X_l}$. This can be viewed as arc consistency reinforcement. A copy of the deleted proposals should be saved for other use in case the meeting $X_l^{A_j}$ is canceled. Finally, the Proposer agent must send the obtained reduced set of possible dates of $X_k^{A_i}$ to all of the Participant agents to ask them to first, adapt it to their convenience, and then *rank*[8] the remaining possible dates according to their preferences.

The main objective of this heuristic is to define an ordinal relationship between all the proposals for each meeting according to users' interests. We can thus especially avoid the classical problem in constructing a common interpersonal utility function, which is how to compare preferences not relying on the same preference scale. Each Participant agent that has received the message containing the reduced set of possible dates, starts first by reinforcing node and arc consistency, then by ranking the obtained slot times according to its preferences (from the most preferred to the less preferred date). The higher an agent ranks a particular date, the more point that date will receive. Specifically, a date is awarded one point for each rank below it.

Finally this agent must return the obtained set of possible dates to the Proposer agent. However, if the set of possible dates for a meeting becomes empty, this Participant agent must send a message to the Proposer agent to inform it about the non-possible meeting. We should emphasize the fact that during this step all the agents try to look ahead for already

[8]This is used as a heuristic to decrease the number of BT and consequently the amount of exchanged messages and hopefully speed up the whole solution process.

Algorithm 8 Start message executed by a User agent A_i for each meeting $X_k^{A_i}$

1: Delete from $D_h^{A_i}$ all non-viable values;
2: **if** ($D_h^{A_i} = \emptyset$) **then**
3: Change meeting possible dates for $X_k^{A_i}$;
4: **else**
5: **for all** each $X_l^{A_j}$ such that ($X_l^{A_j}$, dt_p) $\in$ $Calendar^{A_i}$ and $W_{X_l} > W_{X_k}$ **do**
6: Delete($D_k^{A_i}$, dt_p);
7: Add($DReserve^{A_i}$[k], dt_p);
8: **if** ($D_k^{A_i} = \emptyset$) **then**
9: Change meeting possible dates for $X_k^{A_i}$;
10: **else**
11: **for all** $A_j \in Part(X_k^{A_i})$ **do**
12: *Send*(*self*, A_j, RedMeetCalendar: $D_k^{A_i}$ with:W_{X_k});
13: **end for**
14: **end if**
15: **end for**
16: **end if**

scheduled meetings while reinforcing arc-consistency; this is in order to avoid maximum backtracking[9] in the next step. Otherwise, the first step is finished and the second step can be started.

The Proposer agent tries first to find the proposal that maximizes the utility of the meeting, and then sends it to the concerned acquaintances. To compute the utility of each proposal, the Proposer agent creates a pseudo common-scale based on the obtained ranking and then computes the utility of each possible meeting time and choose the proposal according to the optimality criteria described in Section 4. Each agent A_j that has received a proposal for a meeting $X_k^{A_i}$ must check whether it can still accept it or not. In the case of conflict, i.e., the agent A_j has meanwhile received a proposal for another meeting $X_l^{A_m}$ at the same time as the meeting $X_k^{A_i}$, the agent should act as follows:

- If $W_{X_k} < W_{X_l}$, it must send a negative answer to the Proposer agent A_i and ask it to relax its proposal.
- Otherwise, in the case of $W_{X_k} > W_{X_l}$, the agent must proceed in two steps: first, send a positive answer to the Proposer A_i and second send a message to the Proposer agent A_j of the meeting $X_l^{A_j}$ to invite it to relax its preferences for this meeting.
- Finally, in the case where $W_{X_h} = W_{X_l}$ (case of conflict between two meetings), the

[9]To avoid that the proposer choose a date and come back on it in case of non-availability of at least one participants

agent will try to apply one of the three proposed issues (section 3), i.e., choose always the best, choose one of the meetings at random or apply the *metropolis* criterion, to decide which agent should relax its preferences.

Accordingly, each agent that has proposed a meeting and received at least one negative answer must change its proposal. The same process resumes until an agreement is reached among all of the participants or until testing all of the solutions,no agreement has been reached. In the latter case, the Proposer agent must inform the participants that the meeting is canceled. Note that this dynamic allows a premature detection of failure: absence of solution for a meeting. This in the case when the set of possible dates of the concerned meeting becomes empty.

8.3.3 Process of meetings alterations

In real-world applications MS problems are subject to many changes, defined on one side by the arrival of new, more important meetings (especially when all the other meetings have been already approved) and on the other side by the cancelation of one (or more) meeting(s) which, can lead to the possibility of scheduling other meetings, previously detected as non-possible. Therefore, we have used an incremental approach that can handle all forms of alteration in the system without restarting the solving process from scratch. In the following, we present the behavior of our protocol in the case of restrictions and relaxations.

The restrictions

For each new arrival meeting $\mathrm{X}_k^{A_i}$ with the priority $W_{X_k}^{A_i}$, the Proposer agent must first eliminate non-viable values from the domain of this meeting by enforcing node and arc consistency and secondly, must send the obtained slot times to the participants. At the end of this step, and after receiving all the answers from the participants, the agent must choose the date that maximizes the global utility of meeting $\mathrm{X}_k^{A_i}$. If this date is used by another less significant meeting, $\mathrm{X}_l^{A_j}$, where $W_{X_k}^{A_i} > W_{X_l}^{A_j}$, then the latter meeting $X_l^{A_j}$ must be changed. Therefore, the agent A_j (the Proposer of $\mathrm{X}_l^{A_j}$) should be invited to relax its preferences. The proposed date must be communicated to all the participants. Each one of them must check the date and reply to the Proposer, and the same dynamic resumes until the system reaches its temporary stable equilibrium state (because of the dynamics of the system).

We must note that in the worst case, all meetings $\mathrm{X}_l^{A_j}$ with lower priority will be relaxed and the system will stop temporarily with the schedule of the meeting having the lowest priority. The revision of all the decisions to fix a new meeting (when adding a new meeting with highest priority) is slightly unrealistic. Then, in our system we propose applying a penalty (a decrement in the priority of the new meeting) according to the number of involved meetings that must be rescheduled at each step. The main goal of this new process is to speed up the search for the new optimal solution.

The relaxations

For each canceled meeting $X_k^{A_i}$, the concerned agent must check if it can increase the utility of another already scheduled meeting (one or more by propagation). To achieve this goal, this agent must first examine all meetings $X_l^{A_j}$ (called candidate meetings) with $A_i \in$ *Part*($X_l^{A_j}$) and $W_{X_k}^{A_i} > W_{X_l}^{A_j}$ (starting with the most important candidate meeting). In other words, the agent A_i can ask the participants in $X_l^{A_j}$ for further negotiation, if it realizes that this meeting can be held in the date that was taken by the canceled meeting. Nevertheless, this process may increase the utility of some (or all) candidate meetings. This protocol may also allow some meetings, which had been checked as non-possible, to be scheduled (e.g., some meetings that could not be scheduled before may become possible). In addition, we can assume a certain threshold for the meetings to be rescheduled and this is in order to avoid the need to reschedule all meetings in the worst case.

8.4 Example of algorithm execution

Figure8.7(a) shows a simple example of a meeting-scheduling problem formed by three users (A_1, A_2 and A_3) and two new meetings to schedule ($m_i^{A_1}$ and $m_j^{A_2}$). This example is meant to illustrate the dynamic search process for the best proposal and the required messages exchanged among all the participants. Assume that all the users should participate to both new arrival meetings. Each user has its own calendar with its preferences in which, the white boxes represent its availability. While, the gray boxes represent its non-availability. Each agent has also its calendar for the already scheduled meetings with their degree of importance. Both agent A_1 and A_2 have to schedule their meetings. Therefore, both agents begin by concurrently enforcing local consistency on the possible dates of their meetings. Agent A_1 will remove {(M, 5); (W, 4); (Th,5) and (Th, 6)} from the possible dates of $m_i^{A_1}$ and sends the obtained new time-slot with the importance degree of the corresponding meeting (W_{m_i}=0.58) to A_2 and A_3. Agent A_2 will go also through a similar execution.

Next, each agent that received the possible dates from its acquaintances (see 8.7(b)), will start by enforcing local consistency. Second it will rank the remaining possible dates and return them to the sender. For example, A_3 will remove all the times that correspond both to its non-availability and to each already less important fixed meeting, W_{m_k}<0.58 (*resp*. W_{m_k}<0.78) from the received time-slots received from A_1 (*resp*. A_2). The agent A_3 will obtain for $m_i^{A_1}$ {(M, 4); (W, 5); (Tu, 6) (Tu, 7); (W, 6); (F, 1); (F, 2) and (F, 3)} and for $m_j^{A_2}$ {(M, 5); (M, 6) (Tu, 3); (Tu, 4); (W, 5); (W, 6)}. It will rank each set depending on its preferences (for example for $m_i^{A_1}$ {(W, 6); (F, 3); (W, 5); (Tu, 7); (F, 2); (Tu, 6), (M, 4); (F, 1)}) and will return them to the senders.

Let us consider the agent A_1, it will receive the ranked sets from A_2 and A_3. It starts first by computing their intersection of the three sets , calculates second, the ordinal local utility (see 8.4) of each remaining possible time ($U(m_i^{A_1}$, (W, 5))=3+4+3=10; $U(m_i^{A_1}$, (M,

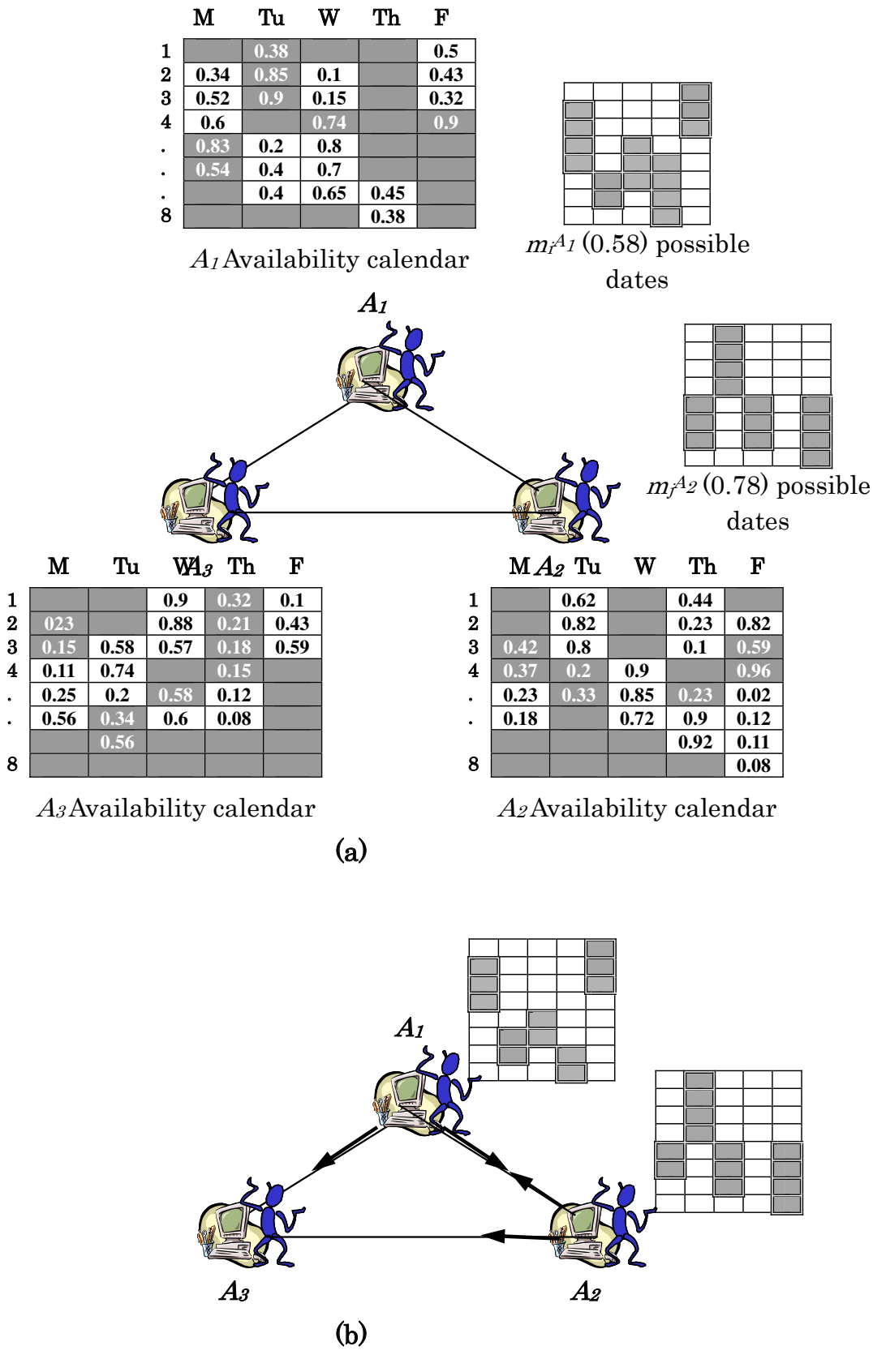


Figure 8.2. Example of the meeting scheduling problems with three users.

Table 8.4. Different time proposals for meeting $m_i^{A_1}$ ranked according to users' preferences.

	A_1	A_2	A_3
4	(W, 6)	**(W, 5)**	(W, 6)
3	**(W, 5)**	(F, 2)	**(W, 5)**
2	(M, 4)	(W, 6)	(F, 2)
1	(F, 2)	(M, 4)	(M, 4)

4))=2+1+1=4; $U(m_i^{A_1}$, (W, 6))=4+2+4=10; $U(m_i^{A_1}$, (F, 2))=1+3+2=6) and finally chooses the date satisfying the mentioned before criteria (see section 3). In this case, A_3 will choose d_p=(W, 5) to enforce preferences similarity among the participants. The proposer agent should receive an agreement for this proposer, from all the participants to confirm it.

Assume that, in the meanwhile, A_2 has just received same proposal (same time) for another meeting $m_k^{A_p}$ with the same importance (W_{m_k}=W_{m_i}=0.58). The agent A_2 will apply the metropolis criteria (see section 3) to decide which meeting has to be rescheduled ($m_i^{A_1}$ or $m_k^{A_p}$) and informed the concerned agent by its decision.

8.5 Evaluation

8.5.1 Theoretical evaluation

Termination

The MSRAC process stops temporarily (dynamic system) when the system reaches a stable equilibrium state. In this state all the agents are temporarily satisfied. An agent is satisfied when it has no meetings to schedule or when it has received all the confirmations from all the other Proposer agents. We assume that between *t* and *t'* there is no new (*resp.* cancelled) meeting. Thus, at time *t*, the number of meetings to be fixed is limited and finite, so the proposed approach stops after making at most this many meetings. We assume that MSRAC approach goes into an infinite loop. This may happen in two cases:

1. While scheduling a meeting.
2. While rescheduling a meeting.

For the first assumption, to schedule a meeting $X_l^{A_i}$ all the participants will cooperate together to find the best date for this meeting. The system will go into an infinite loop while scheduling $X_l^{A_i}$ if and only if the Participant agents reprocess the checked dates (cycle) when

no solution is found. However, the number of possible dates per meeting is discrete and finite. In addition, every checked date is removed from the system to avoid a return to it later. The system will stop when a "good" date if found or when all possible dates for $X_l^{A_i}$ are processed and no possible solution has been found. Therefore our assumption is not true.

For the second assumption, the system goes into an infinite loop if the rescheduling of $X_l^{A_i}$ leads to the rescheduling of $X_p^{A_j}$ and the rescheduling of $X_p^{A_j}$ leads the same to the rescheduling of $X_q^{A_k}$ and finally the rescheduling of $X_q^{A_k}$ leads to the rescheduling of $X_l^{A_i}$. However the rescheduling of $X_l^{A_i}$ leads to the rescheduling of $X_p^{A_j}$ if and only if $W_{X_l} > W_{X_p}$ and the same for the other meetings[11]. Therefore $W_{X_l} < W_{X_p}$ means that the reschedule of $X_q^{A_k}$ will never lead to the reschedule of $X_l^{A_i}$. This contradicts our assumption.

We have to note that the satisfaction state of all the agents in a distributed system can be detected by taking a snapshot of the system, using the well-known Chandy-Lamport algorithm [19]. The termination occurs when all agents are waiting for a message and there is no message in the transmission channels. The cost of the termination process can be mitigated by combining snapshot messages with our protocol messages.

Complexity

Let us consider the complexity of adding a new meeting into an existing schedule. The corresponding MS problem involves *n* for total number of users, *d* for the maximal number of possible dates for each meeting and *c* for the total number of preferred dates for each user. The total number of agents in this system is *n* the same as the total number of users. Suppose that each meeting involves *n* attendees and each user has *m* already scheduled meetings in the calendar. Our approach is composed of two steps.

- In the first step, each agent performs O(*cd*+*md*+*dLog(d)*) operations to reinforce node and arc-consistency and to rank the remaining dates. The Proposer agent then determines the intersection of the received sets of possible dates leading to $(n\text{-}1)d^2$ operations, and the utility of each dates with O(*nd*). Thus, the temporal complexity of this step in the worst case is
  $O(n(cd+md+dLog(d))+(n\text{-}1)dt^2+nd)$.

- In the second step, in order to compute the cost of rescheduling, we assume that at each step the chosen date leads to the rescheduling of one meeting (at most[12]) in the worst case. This leads to *m* successive iterations. Each agent checks its calendar *m* and sends its answer to the proposer in order to choose a new value. This process requires

[11] The rescheduling of a meeting leads to the rescheduling of another meeting if and only if the first meeting is more "important" than the second.

[12] Because we assume that all the users are participants in all meetings

O(m(nm+d)) operations in the worst case. The space complexity, for all the agents, is O(n(d+m)).

Message passing optimality

In order to show that our approach requires the minimum amount of messages passing to reach an agreement among all the attendees, let us assume that we have n agents $\{A_1, \ldots, A_n\}$, each has an already scheduled meeting $X_1^{A_i}$ at the date d_l with $l \in \{1, \ldots, k\}$. The total number of scheduled meetings in the whole system is k at the dates $\{d_1, d_2, \ldots, d_k\}$.

Suppose that the agent A_j proposes a new meeting $X_2^{A_j}$ involving all the agents of the system. The possible dates for this meeting are $\{d_1, d_2, \ldots, d_k, d_{k+1}\}$. According to the mast of the approaches proposed in the literature (including [41, 101, 42]), a proposal d_h ($h \in \{d_1, d_2, \ldots, d_k, d_{k+1}\}$) is selected by agent A_j and passed to all the other agents. Each agent which receives this proposal, replies to the proposer only with a rejection or an acceptance. The same process resumes with another proposal given by another agent[13]. In this case *at least* one agent will reject the proposal. Therefore to reach an agreement among all the participants, this process requires at least *2n(k+1)* messages.

With MSRAC, the proposer sends all the possible candidate dates for a meeting to the participants. Each participant receiving this message will first reinforce arc consistency on the received possible dates in order to avoid as much fruitless backtracking as possible in the next steps. It then ranks the obtained set and sends it only to the Proposer, which determines the intersection of the received sets and obtains the agreement among all of them. Thus the number of required messages in MSRAC is *2n*. Note that receiving all the candidate meeting dates from participants may reveal some information about their local calendars (loss of some privacy). However the only information that Proposer agent A_i may deduce from a participant A_j is its non-availability for some dates. The non-availability of an attendee is due to many different reason, such as another meeting, a business trip, a vacation, personal preference, etc. The proposer cannot reveal the reason of the rejection of the candidate date but he may slowly collect more knowledge by asking for the same date or nearby dates. This is also a common problem for the other approaches. In the worst case the same amount of information will be revealed by all the approaches. Therefore, in order to decrease privacy loss for our approach, we propose to *hide* the identity of the sender. The Proposer agent will then get answers from the attendees without knowing to whom each answer belongs.

8.5.2 Experimental comparative evaluation

To evaluate the proposed MSRAC approach, we have developed the Multi-Agent dynamic with Actalk, an object-oriented concurrent programming language using the Smalltalk-80 environment. In our experiment, we generated random meeting problems. The parameters

[13] we assume that each agent has n possible proposals.

used are: n agents in the system, m meetings per agent, p participants in each meeting, D global calendar, d percentage of possible dates per meeting, w_c weights for the soft constraints, W_{X_l} weight of the event X_l (the weight of each hard constraint is equal to 1) and c the control parameter.

We carried out three kinds of experiments to test the proposed approach. The main goal of the first experiment was to evaluate the efficiency of the three issues proposed to be used in case of conflict (Section 3). We used three versions of the approaches: i) MSRAC-1, the deterministic approach in which each agent always chooses the meeting that increases its local utility (*LU*); ii) MSRAC-2, a non-deterministic approach in which each agent randomly chooses the meeting to schedule on the conflicting date; MSRAC-3, a non-deterministic approach in which each agent apply the metropolis criteria to solve the conflict.

We generated random instances with different numbers of meetings to schedule, in order to vary the number of possible conflicts that may occur, with n =10, m = $\{5, 8, 10, 15\}$, p = 6, D = 50, i.e., each meeting starts between 8AM and 6PM, from Monday to Friday and is one hour long, $w_c \in [0..1]$, $W_{X_l} \in [1..20]$[14], d = 50, $|C_h|$ = 10 and Tp=10. The total number of meetings per instance is, respectively 50, 80, 100, 150. Each instance is executed 30 times. For MSRAC-3 the initial temperature Tp is decreased slowly at each run.

Figure8.3 shows that for the most part, MSRAC-3 is closer to MSRAC-1. MSRAC-2 oscillates more especially when the number of conflicts increases (Figures 8.3c2 and 8.3d2) leading to a great deterioration in the result (in Figure8.3c1 the number of scheduled meetings vary from 34 to 44). This difference can be justified by the fact that MSRAC-3 accepts a deterioration of the *LU* (accept a meeting with lower *LU*) only when the difference in *LU* between the two conflicting meetings is small, with MSRAC-2 the selection of the meeting is totally random which may also increase the number of conflicts. Notice that the number of generated conflicts for MSRAC-3 is almost always the same for all the runs, while there is a big variation for MSRAC-2 (Figures 8.3a2, 8.3b2, 8.3c2 and 8.3d2). Hence, using metropolis criterion to solve the conflict might be more appropriate than random choice and may lead to a better solution than the deterministic approach MSRAC-1 (Figures 8.3b1 and 8.3c1).

In the second kind of experiment, we used two approaches with MSRAC-3, which we will call MSRAC: Asynchronous Backtracking [107] (ABT) and Tsuruta's approach [101].

Recall that, the ABT algorithm is used as a witness approach to appraise the correctness of the results obtained with our approach. As mentioned in Section 1, ABT is a generic and complete algorithm for solving non-dynamic distributed constraint satisfaction problems. Therefore, for this algorithm all the applied problems are treated as static instances. Each agent in the system maintains one variable of the instance. The agents are ordered according to the degree of importance of the variables, i.e., degree of importance (W_{X_l}) of the underlying meeting X_l. The variables (meetings) sharing the same constraint (at least one same participant) are linked together. The approach in [101] presents some restrictions: on the

[14]to increase the probability of having several meetings with same degree of importance.

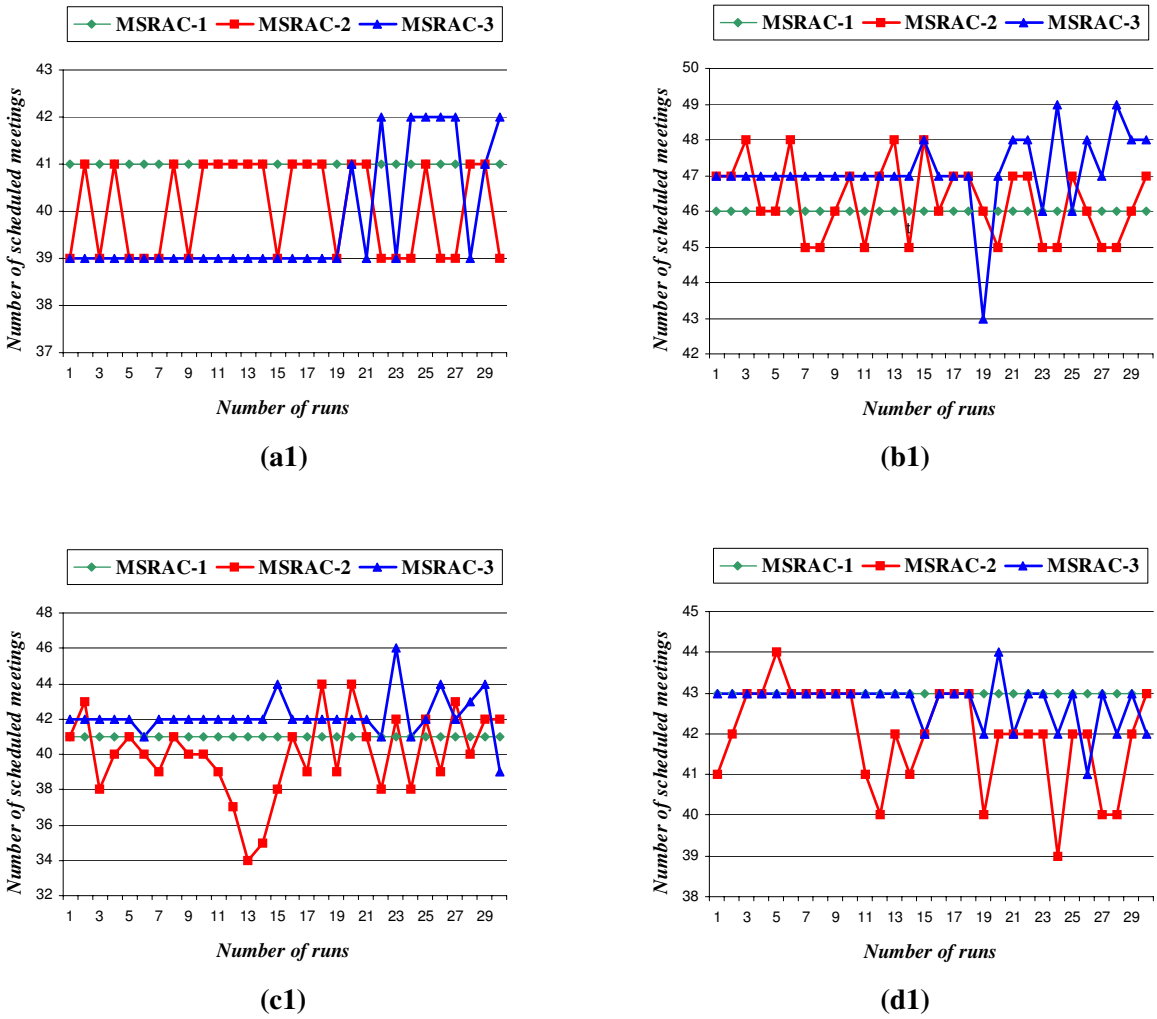


Figure 8.3. Results obtained by the three approaches in mean of number of scheduled meetings (a1, b1, c1 and d1).

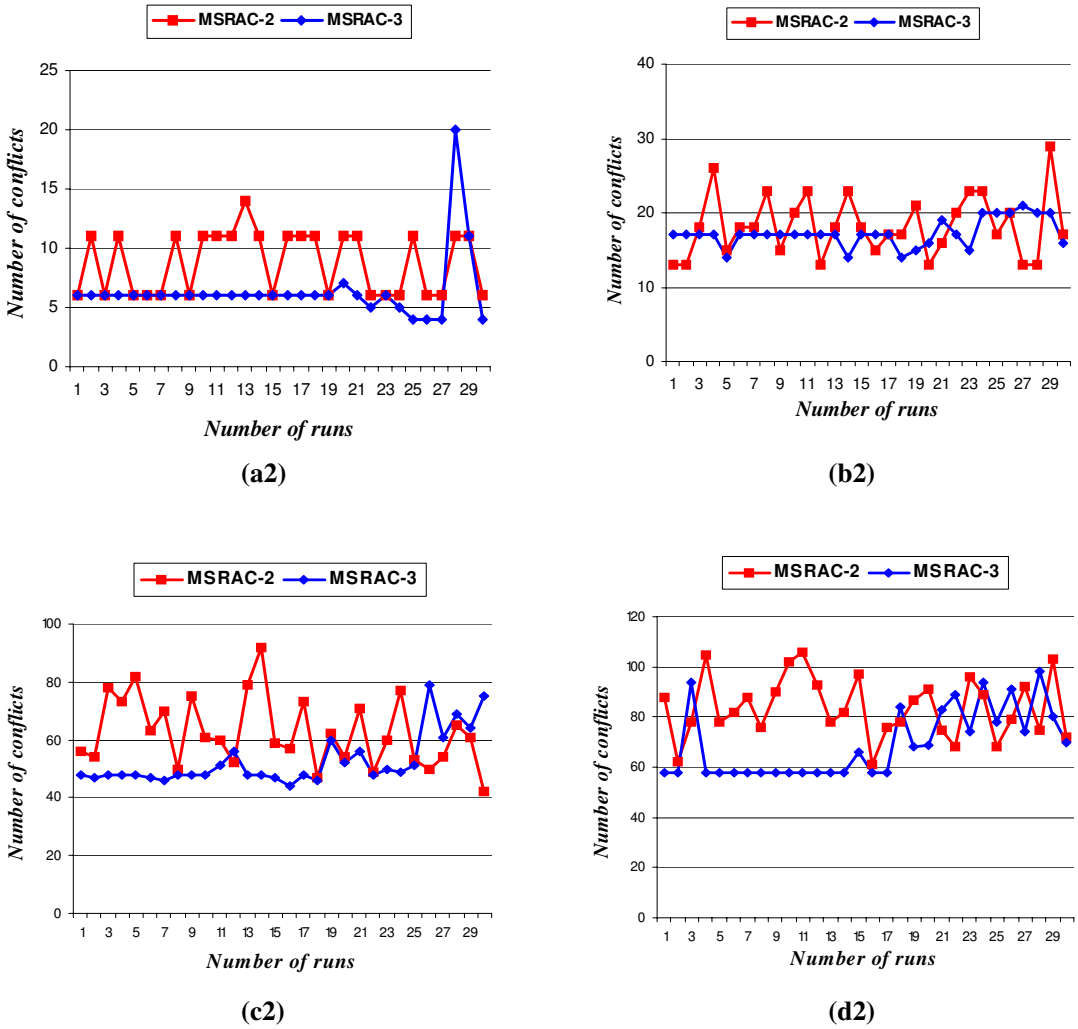


Figure 8.4. Results obtained by the three approaches in mean of number of generated conflicts corresponding to the previous graphs(a2, b2, c2 and d2).

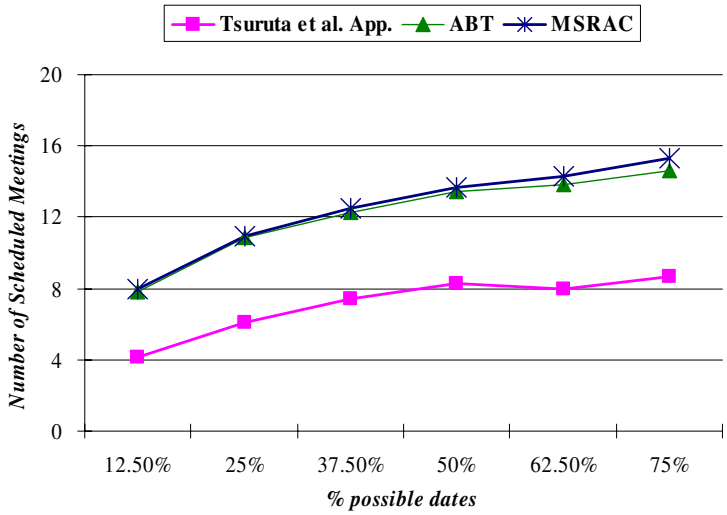


Figure 8.5. Results obtained in mean of number of scheduled meetings.

one hand, the handle of the hard constraints (i.e., all the constraints could be relaxed by this approach) and on other hand, the discrimination between meetings. This approach independently processes all the proposed meetings without regard to their importance to either of the proposer or the attendees.

However in the real world, meetings are not equivalent. Our approach tries then, in its solving process (second step), to schedule the most important meeting maintained by each agent first (unlike the approach in [101]). For this purpose, instances including hard constraints are randomly generated with n =10, m = 5, p = 8, D = 40, $w_c \in$ [0..1], $W_{X_l} \in$ [0..1], d $\in \{12.5\%, 25\%, 37.5\%, 62.5\%$ and $75\%\}$, $|C_h|$ = 10 and c=50. The total number of meetings per instance is 50. For each d we generated 35 instances, then measured the average of the results.

These results are expressed in terms of five criteria: the CPU time (in milliseconds), the number of scheduled meetings, the importance of the meetings, the measurement of real global utility, and the number of exchanged messages. Notice that the first three criteria allow us to especially measure the efficiency of MSRAC. To this end, we have introduced some modifications to the approach in [101] to make it worthwhile for both hard and soft constraints. We carried out the three approaches on the same generated examples using the same parameters.

To simulate a dynamic environment, at each time t each agent knows only about *one* of its meetings (an arbitrary one from its m meetings) and either schedules it or declares its failure to find a solution for it. Once finished the agent will receive a new meeting (another one chosen arbitrarily from the remaining meetings) with higher or lesser importance to process. Every new meetings may lead to the rescheduling of another scheduled one (depending on its importance and the candidate date that will be chosen). Hence at each time t, 1 or n new

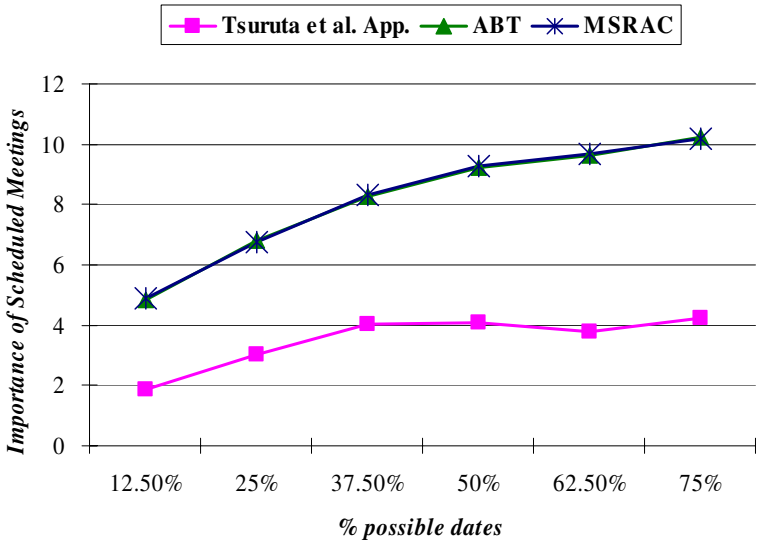


Figure 8.6. Results obtained in mean of the importance of the scheduled meetings.

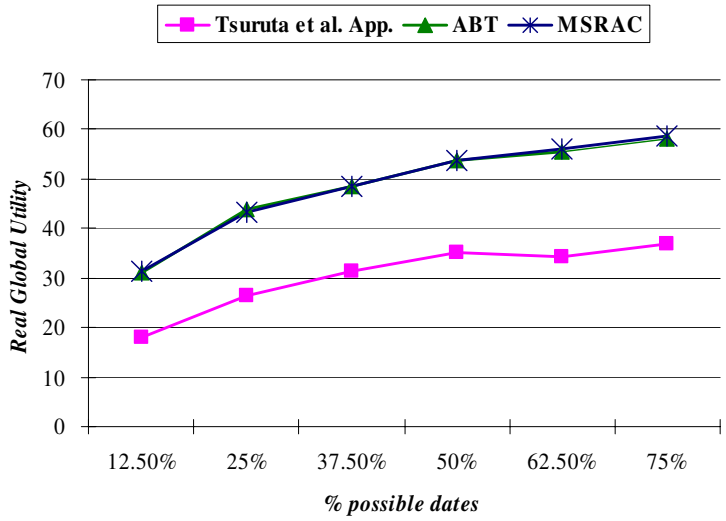


Figure 8.7. Results obtained in mean of the real global utility.

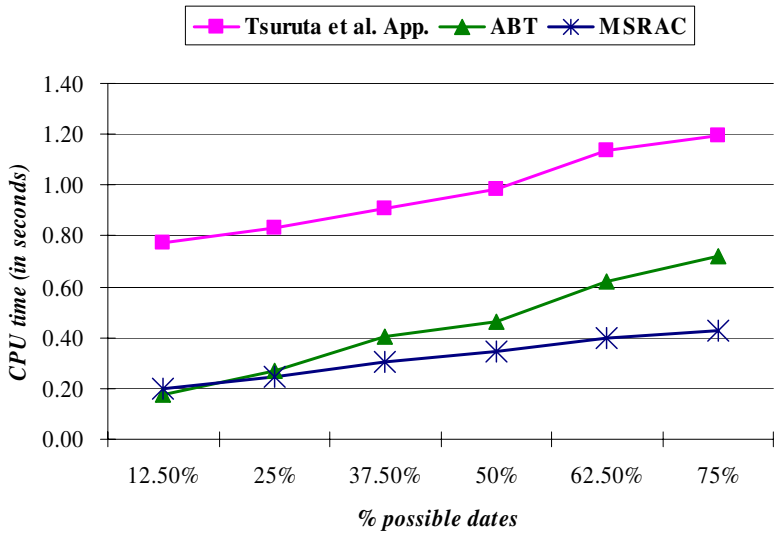


Figure 8.8. Results obtained in term of CPU time.

meetings might be added to the system according to the time required to process the previous ones.

The obtained results show that the MSRAC approach requires, in the majority of cases, less CPU time than the other approaches (Figure8.8), while the CPU time needed by the approach in [101] is about three times more than that needed by our approach. This is can be elucidated by the fact that the first step (reinforcement of local consistency) is useful in order to discard the dates that cannot be in any solution and consequently to avoid exploiting them in the solving process, which leads to CPU time consumption. Let us consider the case of over-constrained instances (possible dates less than or equal 25%)., Figure8.8. shows that ABT requires less CPU time than MSRAC. The main reason is that in such instances, the number of conflicts between meetings is high which may lead to the augmentation of the number of rescheduled meetings. For ABT on the other hand, there is no conflict between meetings; the whole problem, the number of all the possible meetings that may occur in the system is static and known in advance.

As for the number of scheduled meetings (Figure8.5.), ABT and MSRAC schedule almost the same number of meetings. while the Tsuruta approach schedules fewer meetings than the other two approaches. This result shows the efficiency of MSRAC. The small difference noticed in the number of results given by ABT and MSRAC can be justified by the fact that MSRAC uses the *metropolis criterion* in case of conflict. Thus the final result depends on the decision taken towards conflicting meetings. Nevertheless, both approaches provide the same results for the degree of importance of the scheduled meetings (Figure8.6) and the same real global utility (Figure8.7).

In the case of over-constrained problems (d=12,5%), ABT requires fewer exchanged messages than our approach (Figure8.9.). This can be justified by the fact that for this kind

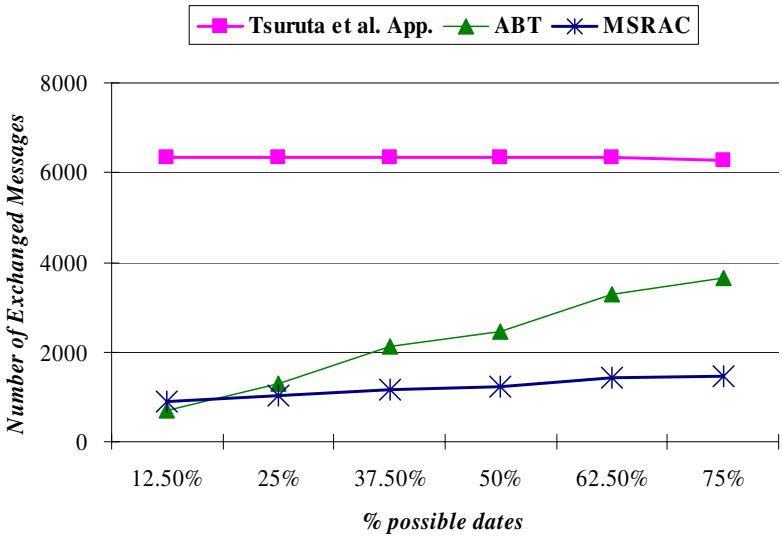


Figure 8.9. Results obtained in term of exchanged messages.

of problem, the agents in ABT can discover merely the absence of solutions due to the low number of possible dates to check. While this number increases, the total number of exchanged messages increases also. With our approach the number of conflicts increases for over-constrained problems, leading to more rescheduling and consequently to more exchanged messages. It is remarkable that the observed difference in the number of exchanged messages between ABT and MSRAC for over-constrained problems is negligible, while the approach in [101], requires more than three times the number of exchanged messages needed for our approach.

In order to appraise the fairness of the above results, we conducted statistical testing, using dependent two samples t-test, to determine whether MSRAC approach and ABT approach could have same mean in terms of CPU time and number of scheduled meetings. The formalization of the null hypothesis and the alternative hypothesis is given in Table 8.5.

Table 8.5. Formalization of the null hypothesis and the alternative hypothesis for both CPU time and number of scheduled meetings.

CPU time	Number of scheduled meetings
H_0: $\mu_{CPU\ MSRAC} = \mu_{CPU\ ABT}$	H_0: $\mu_{NbMt\ MSRAC} = \mu_{NbMt\ ABT}$
H_1: $\mu_{CPU\ MSRAC} < \mu_{CPU\ ABT}$	H_1: $\mu_{NbMt\ MSRAC} > \mu_{NbMt\ ABT}$

The means are measured using Matlab6.1 using significant level $\alpha = 0.05$ and 34 as degree of freedom. For the means in CPU time, Table 8.6 reports the obtained results of each *d*, i.e., percentage of possible time slots for each meeting. According to these results, H_0 is accepted

only for the first case (d=12.5%) with significance equal to 0.997 which means that for this case by chance we would have observed values of T more extreme that the one in this samples in 997 of 1000 similar experiment. The high significance show that in most cases the CPU time required by MSRAC is less than that required by ABT approach. A 95% confidence interval on the mean is [-Inf, 38.3583]. Regarding the other cases, H_0 is rejected with low significance, varying from 0.0114 to 6.36E-12, which means that in all these cases the mean of MSRAC in term of CPU time is almost always less than the mean of ABT approach. The probability to have grater values of T is very low. A 95% confidence interval on the mean is becoming more and more small for high percentage of possible time slots.

Table 8.6. Dependent two samples t-test for the CPU time means of both approaches MSRAC and ABT, for each d.

	Decision	**Significance**	**Confidence Interval**
12.5%	*accept* H_0	0.997	]-Inf, 38.3583]
25%	*reject* H_0	0.0114	]-Inf, -7.4012]
37.5%	*reject* H_0	9.03E-09	]-Inf, -73.7447]
50%	*reject* H_0	5.90E-06	]-Inf, -77.5420]
62.5%	*reject* H_0	2.72E-10	]-Inf,-171.7848]
75%	*reject* H_0	6.36E-12	]-Inf, -231.1478]

For the means in term of the number of scheduled meetings, Table 8.7 indicates that the H_0 is accepted in all cases with high significance for small values of d. For example in case d=25%, the significance $\simeq 0.5$ which means that for this case the probability to observe more extreme value of T is 5 of 10 similar experiments. A 95 % confidence interval on the mean is]-0.8593, Inf] for this case.

To highlight the scalability of our approach, we conducted a second type of experiment in which we tried to increase the size of the problem. We generated 6 groups of random problems. The parameters of the three first groups (groups I, II and III) were: n=10; $m \in \{5, 8, 10\}$; D=50; d=60%; p=7 and $|C_h|$=10. While for the three last groups (groups IV, V and VI): n=20; m=$\{10, 15, 20\}$; D=100; d=60%; p=13 and $|C_h|$=20. We generated 35 instances for each m. Each instance was executed 10 times. Tables 8.8 and 8.9 show the average of the obtained results in term of CPU time and percentage of scheduled meetings for the three approaches.

From these results, we can conclude that MSRAC is scalable, up to 4 times faster than the Sturuta approach (case of $\langle 20; 15 \rangle$ and $\langle 20; 20 \rangle$) and up to 100 times faster than the ABT approach ($\langle 20; 20 \rangle$). For the number of scheduled meetings, MSRAC and ABT planned almost same percentage of meetings, while for Tsuruta approach in [101], the number of scheduled

Table 8.7. Dependent two samples t-test for the number of scheduled meetings means of both approaches MSRAC and ABT, for each d.

	Decision	Significance	Confidence Interval
12.5%	*accept* H_0	0.3465	]-0.5498, Inf]
25%	*accept* H_0	0.4587	]-0.8593, Inf]
37.5%	*accept* H_0	0.2638	]-0.4646, Inf]
50%	*accept* H_0	0.3301	]-0.5894, Inf]
62.5%	*accept* H_0	0.1666	]-0.3234, Inf]
75%	*accept* H_0	0.0872	]-0.1480, Inf]

meetings at each instance is about 50% of that achieved by the two other approaches. Hence, it is noteworthy that our approach seems to be more appropriate to real-world applications by dealing with users' hard constraints and by bringing forward consideration of discrimination among the proposed meetings. In addition, the first step of the proposed approach can prematurely detect the impossibility for reaching any agreement among all the participants.

Table 8.8. Results obtained in mean of CPU time.

	⟨10; 5⟩	⟨10; 8⟩	⟨10; 10⟩	⟨20; 10⟩	⟨20; 15⟩	⟨20; 20⟩
Tsuruta App	1349.91	1933.15	2418.35	22883.18	34086.74	48248.95
ABT App	911.91	3729.94	8845.74	29713.62	90353.53	185200.63
MSRAC	544.94	777.24	936.79	5225.35	7807.74	9551.53

Table 8.9. Results obtained in mean of percentage of scheduling meetings.

	⟨10; 5⟩	⟨10; 8⟩	⟨10; 10⟩	⟨20; 10⟩	⟨20; 15⟩	⟨20; 20⟩
Tsuruta App	28.91%	21.91%	18.44%	8.37%	6.84%	5.20%
ABT App	46.91%	33.86%	28.12%	21.63%	15.94%	12.36%
MSRAC	50.36%	35.18%	29.68%	22.22%	16.51%	12.91%

8.6 Summary

In this chapter, we propose a new scalable and dynamic approach (MSRAC) to solve meeting scheduling problems. In this approach, we tried to integrate the main features of the MS problem such as: user preferences, user non-availability, importance of the meeting, etc. to reflect ideally real-world applications. To this end, we proposed to use two kinds of

constraints to model the users' requirements: hard constraints to model the non-availability of a user and soft constraints to define the user's preferences. Note that the integration of these features transforms the problem into an optimization problem.

The Multi-Agent underlying model associates an agent with each user and makes the agents interact by sending point-to point messages containing only relevant information. Basically, this approach consists of two steps. The first reduces the initial problem by reinforcing some level of local consistency (node and arc consistency). The second step solves the resulting meeting scheduling problem while maintaining arc-consistency. In the proposed protocol, the information shared among all the agents is kept to a minimum without reflecting on the efficiency of the cooperative decision taken by all these agents.

All the meetings can be processed in a parallel and distributed manner, while achieving the meetings' higher utilities. This can be obtained as a side effect of interactions between the agents of the system, while both minimizing the amount of message passing and ensuring the user's privacy. We should note that the underlying protocol forbids only parallel meetings with common participants. The MSRAC approach was compared with the ABT approach [107] and Tsuruta's approach [101]. The obtained results show that our approach is efficient, scales better and performs less message passing for almost the same solutions. This approach has been published in [8, 11] and under reviewing in the *International Journal of Engineering Applications of Artificial Intelligence* [3].

Algorithm 9 Main procedures executed by each Proposer agent A_i

RedMeetCalendar:D **with**:W

1: Delete from D all non-viable values;
2: **if** ($D = \oslash$) **then**
3: *Send*(*self*,*sender*,MeetNotPossible);
4: **end if**
5: **for all** $X_l^{A_j}$ such that ($X_l^{A_j}$, dt_p) $\in$ *Calendar*^{A_i} and $W_{X_l} > W$ **do**
6: Delete(D, dt_p);
7: **if** ($D = \oslash$) **then**
8: *Send*(*self*, *sender*, MeetNotPossible);
9: **else**
10: Rank(D);
11: /* According to A_i preferences $w_p^{A_i}$ */
12: *Send*(*self*, *sender*,Reply: D);
13: **end if**
14: **end for**

Reply: D

1: **if** (All ranked D are received from all $A_j \in$ *Part*($X_k^{A_i}$)) **then**
2: Update($D_k^{A_i}$) /* According to received Ds*/
3: **end if**
4: $dt_p \leftarrow$ the date of *higher utility*; /* The choice of the date should be done according to the two equations 2 and 3 given in section 3[10] */
5: **if** (dt_p = *nil*) **then**
6: **for all** $A_j \in$ *Part*($X_k^{A_i}$) **do**
7: *Send*(*self*, A_j, MeetNotPossible);
8: **end for**
9: **else**
10: **for all** $A_j \in$ *Part*($X_k^{A_i}$) **do**
11: *Send*(*self*, A_j, ReceiveProp:dt_p with:W_{X_k});
12: **end for**
13: **end if**

Algorithm 10 Main procedures executed by each Proposer agent A_i

ReceiveProp:*Prop* **with**:*W*

1: **if** (($\exists$ ($X_l^{A_j}$, dt_p) $\in$ *Calendar*$_{A_i}$ such that *Prop* = d_l and $W_{X_l} > W$)) **then**
2: *Send*(*self*, *sender*, UpdateProp:*Prop* with:*W*);
3: **else**
4: **if** ($\exists$ ($X_l^{A_j}$, dt_p) $\in$ *Calendar*^{A_i} such that *Prop* = dt_l and $W_{X_l} < W$) **then**
5: *Send*(*self*, A_j, UpdateProp:d_p with: W_{X_l});
6: *Send*(*self*, *sender*, MeetingOK);
7: **else**
8: apply one of the three proposed issues (section 3.) to decide which agent should relax its preferences;
9: **end if**
10: **end if**

UpdateProp:*Prop* **with**:*W*

1: Delete ($D_k^{A_i}$, *Prop*);
2: Add(*DReserve*^{A_i}[*k*], *Prop*);
3: **if** ($D_k^{A_i} \neq nil$) **then**
4: $dt_p \leftarrow$ the date of h*igher utility*;
5: **for all** $A_k \in$ *Part*($X_k^{A_i}$) **do**
6: *Send*(*self*, A_j, ReceiveProp: dt_p with: W_{X_k});
7: **end for**
8: **else**
9: **for all** $A_j \in$ *Part*($X_k^{A_i}$) **do**
10: *Send*(*self*, A_j, MeetNotPossible);
11: **end for**
12: **end if**

Chapter 9

Asynchronous Constraint-based Approach: A New-Born in the ABT Family

In this chapter we present a novel constraint-based complete and generic constraint-based approach to solve a CN with any arity. However, as mentioned before most the existing approaches for solving DisCSP are variable-based approaches, allow the adding of new links and especially proceed by recording nogoods in order to ensure completeness. The cost of the search process grows with the number of connections, the amount of nogoods recorded and with the required constraint checks. Only the work of Silaghi (AAS) in [98] is constraint-based approach. It is considered as ABT for dual graph. Nevertheless, this techniques is based on exchanging aggregating ranges of tuples rather than single values. Nevertheless, determining ranges of tuples requires a high amount of constraint checks and consequently may increase the cost of the solver.

The new approach (that we called DisAS for distributed asynchronous search) described in this chapter, is based in a part on a *lazy* version of DRAC protocol, without adding new links and especially without any nogood recording. In addition in this work we propose a generic distributed approach to compute a static ordering, in which we save as many links as possible hopefully to decrease the set of exchanged messages and to make it practically useable.

In this chapter, we present first the constraint-based new approach. Second we give the proposed protocol. Then we discuss the theoretical result. Finally, we illustrate the experimental results followed by a conclusion.

9.1 Constraint-based asynchronous search approach

9.1.1 Multi-agent architecture

The proposed multi-agent architecture is based on the dual representation of a CN. This model involves two kinds of agents: Constraint agents and an Interface agent. The latter agent is added to the system in order to inform the user of the result. Each agent has a simple structure formed by its acquaintances (the agents that it knows), a local memory composed of its static and dynamic knowledge, a mailbox where the agent stores the received messages, and a local behavior.

All the agents communicate by exchanging asynchronous point-to-point messages containing only relevant information. An agent can send a message to another one only if it knows it (it belongs to its acquaintances). For transmission between agents, we assume that messages are received in the order in which they were sent. The delivery time for messages is finite.

9.1.2 Generic parallel new method for static constraint ordering

The complexity of the CN and the number of exchanged messages are highly depending on the existing connections between the agents of the system. In this section we propose a new distributed method to define an optimal global order (i.e., optimal in term of connections) between the agents. In our system, each agent will locally compute its position in the ordering according to its variables. The first variable of an agent A_i, defines its level and will be used to determine both its set of higher level acquaintances, i.e., $Parents^{A_i}$, and its set of lower level acquaintances, i.e., $Children^{A_i}$. The agent A_i responsible of the constraint C_{ij} will be the level i. The obtained graph should satisfy Property4 in order to ensure the completeness of the solving approach.

Property 4 *For each variable $X_i \in X$, for all the agents A_k such that $X_i \in$ Var($C_{ij}^{A_k}$), A_ks' are related through a single and continuous path.*

To illustrate the main principle of this method, we assume initially that for each agent A_i the set of children is all the constraints with which the agent shares at least one variable (basis of the dual graph). Each agent A_i will reduce the set of its children, $Children^{A_i}$, by using the following rules:

Rule 1. Remove all A_l, *Var*($C_{ik}^{A_l}$)=$\{X_i, X_k\}$, from $Children^{A_i}$ (*Var*($C_{ij}^{A_i}$)=$\{X_i, X_j\}$) such that $A_l \prec_{lo} A_i$, i.e., $A_l \prec_{lo} A_i$ if and only if $k < j$.

Rule 2. Remove all A_h, *Var*($C_{fj}^{A_h}$), from $Children^{A_i}$ (*Var*($C_{ij}^{A_i}$)=$\{X_i, X_j\}$) such that $f > i+1$ and there is no $A_l \in Children^{A_i}$ with *Var*($C_{mj}^{A_l}$)=$\{X_m, X_j\}$ and $m \in \{i, \ldots, (f-$

l)}.

Once the set of the children is reduced, each agent A_i will inform each agent $A_l \in Children^{A_i}$ that it is the father. Then each agent A_l receiving the above message will add A_i to its set of parents, $Parents^{A_l}$=$Parents^{A_l} \cup \{A_i\}$.

It is noteworthy that for a full connected graph, using *n* variables, the total number of constraints is n(n-1)/2. For each constraint we will have 2(n-2) links then the total number of links, for the dual graph is O(n^3). In case of an ordered dual graph, each constraint C_{ij} has (2n-(i+j)-1) ordered links. The total number of ordered links is n(n-1)/2*(2n-(i+j)-1). As for our ordering method, each constraint of level $i \in \{2, \ldots, (n\text{-}1)\}$ will be connected to 2(n-i) other constraints in the next level; only the first level i=1 needs more (n-2) ordered connections. Thus the remaining ordered connections for the proposed method is n(n-2). We can easily see that our method saves many more connections and consequently decreases the complexity of the exchanged messages in a real distributed computer architecture. In addition, we assume to designate a leader for each variable, which will be responsible of this variable. Each agent A_i that has no parent for at least one of its variables X_k, it will be the leader of this variable.

Figure9.1 illustrates the proposed distributed static ordering proposed method, where in the upper side a dual constraint graph is represented. To achieve the required order with the minimum connections, each agent performs the algorithm detailed in Algorithm11. Each agent possesses the set of its acquaintances. Once this method is executed, the constraint ordering obtained is the one represented in Figure9.1(b) where:

A_1: $Parents^{A_1}$={}; $Children^{A_1}$={A_3}.
A_2: $Parents^{A_2}$={}; $Children^{A_2}$={A_3, A_4, A_5}.
A_3: $Parents^{A_3}$={A_1}; $Children^{A_3}$={A_5}.
A_4: $Parents^{A_4}$={A_2}; $Children^{A_4}$={A_6}.
A_5: $Parents^{A_5}$={A_2, A_3}; $Children^{A_5}$={A_6}.
A_6: $Parents^{A_6}$={A4, A5}; $Children^{A_6}$={ }.

The gray circles represent the leaders of the variables. The maximal number of leaders is *n*. These agents will be used to perform backjumping (second step) in the solving process. The ordering technique can be performed with a fixed number of messages and all the agents are totally independent. All the agents can perform parallel computations at the same time, leading to a good parallelization feature.

Furthermore, this method can be used to detect the existence of cycles in the CN in a distributed manner. When an agent and its two parents share the same variable, then this agent and its parents form a cycle of three variables. For example, in Figure9.1(b) {A_4, A_5,

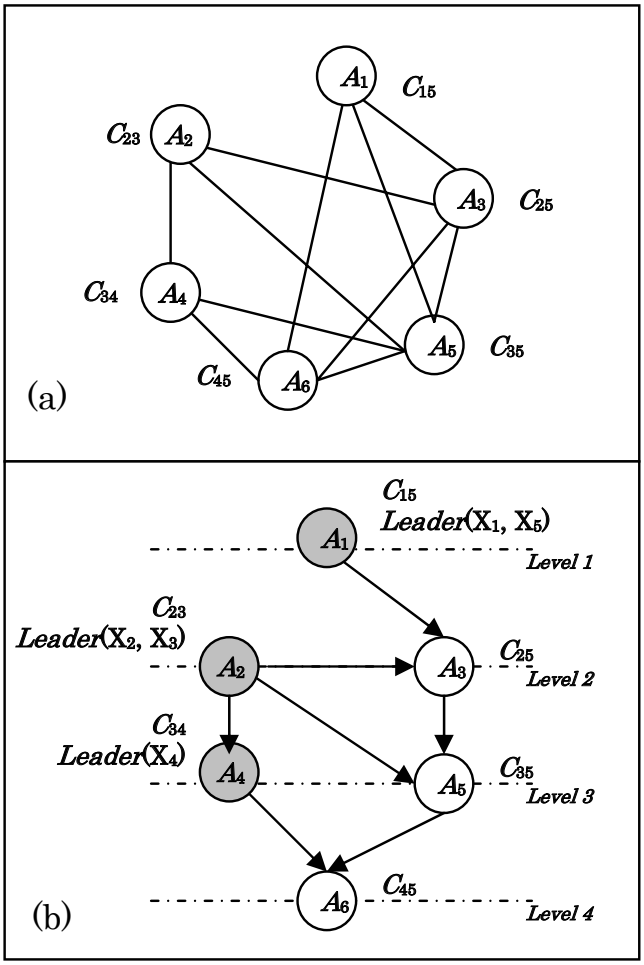


Figure 9.1. Distributed asynchronous constraint ordering.

Algorithm 11 Distributed constraint ordering main process executed by each agent A_i.

begin

1: $Children^{A_i} \leftarrow A_k \in A \,/\, Var^{A_i} \cap Var^{A_k} \neq \varnothing$;
2: $Parent^{A_i} \leftarrow \varnothing$;
3: **for all** $A_k \in Children^{A_i}$ **do**
4: **if** ($A_k \prec_{lo} A_i$) OR (($level(A_k) > level(A_i)$) AND ($\exists$ $A_h \in Children^{A_i}$ such that $A_h \prec_{lo} A_k$ and $Var^{A_i} \cap Var^{A_k} = Var^{A_i} \cap Var^{A_h}$)) **then**
5: $Children^{A_i} \leftarrow Children^{A_i} \ A_k$;
6: **end if**
7: **end for**
8: **for all** $A_h \in Children^{A_i}$ **do**
9: $sharedVar \leftarrow Var^{A_i} \cap Var^{A_h}$;
10: *sendMsg*($Self$, A_k, "$IamYourParent$:$self\ for$:$sharedVar$");
11: **end for**

$A_6\}$ define a cycle, i.e., the two parents A_4 and A_5, of A_6, share the variable X_3.

9.1.3 Solving asynchronous process global dynamic

The main common global objective of all the agents is to solve any constraint problem. This dynamic is divided into two steps:

- First step, a "partial" enforcement of arc consistency [65], consists in pruning some non-viable values and propagating them to higher level agents in order to decrease the amount of backtracking and hence reduce the complexity of the solver. This step can be viewed as a *lazy* version of DRAC approach.
- Second step, the solving process, consists in solving the obtained problem via interactions and negotiations among all the agents of the system. Each agent searches for the suitable tuple that, on the one hand, satisfies its associated constraints and, on the other hand, satisfies all the agents belonging to its parents and children, i.e., $\forall$ A_h such that $A_h \in Parents^{A_i} \cup Children^{A_i}$.

In this protocol, agents are ordered statically and inter-agent links are directed from high priority to low priority agents, for two main reasons, to build an acyclic graph and to ensure a continuous path between agents sharing the same variable (using the aforementioned method).

Each agent has at most two parents and none or many children. Each agent maintains only a short and current view of the values taken by its parents. This view is defined by the tuple *t* chosen by its parent(s). Each agent runs a similar process, and updates the stored information received from its parent(s) in the form of an agent view.

During the first step, each agent A_i enforces lazy arc consistency on the domain of its variables *Var*($C_{ij}^{A_l}$). Each agent seeks the "*first support*" [6] of each value a of its variables. If a valid tuple *t*, such that *t*[index(*Var*($C_{ij}^{A_l}$)), X_j]=*a*, is found, then (*t y*) is stored in the set of first support. Each agent maintains this set in order to avoid redundant checks. The value of *y*=(*i*-1) is added in order to indicate whether *t* is the first tuple support for X_i or not. Each deleted value *a* in D(X_i) should be communicated only to the parents (*lazy* enforcement). The main reason is to minimize the number of exchanged messages and to avoid seeking solution containing these non-viable values which may increase the number of constraint checks.

The same process resumes; each agent that received a non-viable value has to send it to its concerned parents. The deletion process continues until there are no more values to propagate. If a domain of at least one variable becomes an empty set, then this agent has to inform the Interface agent of the inconsistency of the problem in order to stop the system.

The system then moves to the solving process (second step). Each agent A_i in A will choose a tuple *t* from its set of first support, i.e., $firstSupport^{A_i}$. If the agent is a leader of at least one variable, $Leader$(X_i), he has to choose the "first possible viable tuple *t*" in order to guarantee the completeness of the proposed approach and not escape any solution. The agent has to communicate the chosen tuple t to its children $Children^{A_i}$ as a new proposal. Each agent that received a proposal from its parents updates first its set of received proposals, $listProp^{A_i}$ and then tries to adjust its proposal, $Proposal^{A_i}$ according to the ones it receives.

If the agent succeeds in finding a new viable proposal compatible with its current view, then this new proposal has to be communicated to its children. Otherwise, the agent chooses the "nearest" leader of its variables and asks it to change the value of the concerned variable. This jump allows us to speed up the solving process and also to reduce the number of exchanged messages.

The leader has to inform the agent whether or not it can change the value. In the negative case, in which there is no possible other value for the underlying variable, the agent has to ask a second leader before propagating the request to a high priority agent (the leader of the leader). If the head of all the agents receives a request to change its value and he cannot find any more viable tuple, the agent sends an interruption message to the interface to inform it of the non-existence of a solution.

9.2 Illustrative example

9.3 Theoretical analysis

9.3.1 DisAS soundness and completeness

For the correctness of our approach we have to prove the following two propositions:

Algorithm 12 Start Process executed by each agent A_i.

begin

1: **for all** $A_i \in A$ **do**
2: $propState^{A_i} \leftarrow$ false;
3: **if** $|Parents^{A_i}| \leq 1$ **then**
4: Choose the first tuple *t* such that *t* satisfy Const(A_i);
5: **if** $|Parents^{A_i}|$=0 **then**
6: $propState^{A_i} \leftarrow$ true;
7: **else**
8: Choose *t* such that $t \in firstSupport^{A_i}$;
9: **end if**
10: **end if**
11: **end for**
12: $proposal^{A_i} \leftarrow t$;
13: **for all** $A_j \in children^{A_i}$ **do**
14: var $\leftarrow CommonVar(A_i, A_j)$;
15: ind $\leftarrow$ index(Var^{A_i}, var);
16: *sendMsg*(self, A_j, "*ProcessProposal*:proposal^{A_i} *for*: var *at*:ind *withPropState*:propState");
17: **end for**

Propostion 1 *The combination of all the tuples received by the interface agent at the stable state is a non-empty set,* $\forall$ *A*$_i$ *and A*$_j$ *such that* $Var^{A_i} \cap Var^{A_j} \neq \emptyset$, t^{A_i} *and* t^{A_j} *have same value for the shared variable.*

Proof. Assume that the interface agent received two instantiations with different values for the common variable, t^{A_i}=((X$_k$, v_k) (X$_l$, v_l)) and t^{A_i}=((X$_m$, v_m) (X$_k$, v'_k)). The two agents are linked, assume that $A_j \prec_{lo} A_i$. The agent A_i can send its instantiation to the interface agent only, and only if its state is true, i.e., its instantiation satisfies that of its parents and all the states of its children are true (see termination detection conditions). The state of A_j depends on the received instantiation from A_i, i.e., its instantiation should satisfy the one received, otherwise the agent should generate a conflict with state set to false. Then the two received values for the variable X$_k$ should be the same.

Propostion 2 *Every found combination of variables is a solution of the problem.*

Proof. Let consider *S* as a combination of variables' values generated by the Interface agent after receiving tuples from all the agents. Assume that *S* is not a solution for the problem, there exists C$_{ij}$ $\in$ C such that *S* does not satisfy C$_{ij}$. Assume that this constraint is

Algorithm 13 Main messages exchanged by the agents of the system.

ProcessProposal:*prop* **for**:*shVar* **at**:*ind* **withPropState**:*myState*

```
1:  add(listProp^{A_i}, prop);
2:  add(statePr^{A_i}, mySate);
3:  if (|parents^{A_i}| = 1) then
4:     if (proposal^{A_i} [index(A_i, shVar)]= prop[ind]) then
5:        for all A_j ∈ children^{A_i} do
6:           var ← CommonVar(A_i, A_j);
7:           ind ← index(Var^{A_i}, var);
8:           propState^{A_i} ← true;
9:           sendMsg(self, A_j, "ProcessProposal:proposal^{A_i} for:var at:ind withPropState:propState^{A_i}");
10:       end for
11:    else
12:       /* value of shared variable not the same*/
13:       InconsistentValueFor:prop at:ind
14:    end if
15: else
16:    /* A_i has two parents*/
17:    CycleConflictFor:prop at:ind
18: end if
```

maintained by the agent A_i. If *S* does not satisfy C_{ij} this means that the instantiation generated by A_i is not compatible with its parents. Then the state of the agent cannot be true and then this agent cannot send its instantiation to the interface, which contradicts our assumption.

As for the completeness of the proposed approach, the directed used dual graph has no cycle. Also, every agent that is a leader of at least one its variables cannot return to a already chosen tuple; this agent always tries to go ahead in order to avoid arriving at the same conflict. Therefore, in case of backtracking, i.e., the tuple $t_1^{A_i}$ is inconsistent with at least one agent of lower priority, A_j will choose $t_2^{A_i} \succ t_1^{A_i}$. The other agents may consider any consistent tuple because their instantiation depends on that of their two parents.

The absence of solution can be detected during the enforcement of *lazy* arc consistency (during the first step), or in the case where a agent cannot find a consistent instantiation even after performing a local exhaustive search, by asking its leaders to provide more values. In this case, the agent will inform the interface to stop the whole system and communicate the non-existence of a solution to the human user.

Algorithm 14 Main procedure to process inconsistent value.

InconsistentValueFor:*prop at*:*ind*

1: *tuple* ← *searchFirstTupleIncludes*:prop[ind];
2: **if** (*tuple* = nil) **then**
3: /*no viable tuple is found*/
4: $propState^{A_i}$ ← false;
5: **if** ($myState$ = true) **then**
6: /* his parent not in conflict*/
7: $progLevel^{A_i}$[shVar] ++;
8: *sendMsg*(self, $Leader^{A_i}$[shVar], "*moreValueFor*:shVar *not*:prop[ind]");
9: **else**
10: /* Ai parent's did not take his final decision the prop might be changed*/
11: **end if**
12: **else**
13: $proposal^{A_i}$ ← *tuple*;
14: $propState^{A_i}$ ← true;
15: **for all** $A_j \in children^{A_i}$ **do**
16: var ← $CommonVar$(A_i, A_j);
17: ind ← index(Var^{A_i}, var);
18: $sendMsg$(self, A_j,"*ProcessProposal*:$proposal^{A_i}$ *for*:var *at*:ind *withPropState*: $propState^{A_i}$");
19: **end for**
20: **end if**

9.3.2 Termination

Most of the termination processes of the existing MAS approach are based on the well-known algorithm of [19]. This algorithm requires the taking of snapshots of the system at different stages leading to an increase in the number of exchanged messages. In our work, we propose that the stable state will be detected progressively by the agents of the systems.

The main idea consists of defining a state for each agent A_i, this state is set to true if A_i and all its children, i.e., $\forall$ A_l / $A_l \in Children^{A_i}$, succeed in instantiating their variables. The detection process will be detected by the leaves of the graph, i.e., all the agents A_l such that $Children^{A_i}$=$\emptyset$, and will be progressively propagated to the head(s), i.e., all the agents A_l such that $Parents^{A_i}$=$\emptyset$, of the graph to be announced to the Interface agent. Each agent A_l that has no child and succeeds in instantiating its variables will set its state to true, $State^{A_i}$=*true*. Then A_l will inform its parents by its state.

The acyclic structure of the graph allows us to avoid entering an infinite loop and consequently to gradually detect the final state. To summarize, if the head of the graph receives

Algorithm 15 Main procedure to process a cycle conflict.

CycleConflictFor:*prop at*:*ind*

1: **if** $|listProp^{A_i}| > 1$ **then**
2: **if** $\neg consistent(listProp^{A_i})$ **then**
3: /* A_i has to wait because one of the two parents will change its proposal */
4: **else**
5: *tuple* $\leftarrow generateTuple(listProp^{A_i})$;
6: **if** (*tuple* $\neq proposal^{A_i}$) AND (*tuple*$notin\ firstSupport^{A_i}$) AND (*tuple* Not Satisfy Const(A_i)) **then**
7: /* the nearest Leader to A_i */
8: $propState^{A_i} \leftarrow$ false;
9: $progLevel^{A_i}$[1] ++;
10: *sendMsg*(self, $Leader^{A_i}$[1], "*moreValueFor*:sharedVar *not*:prop[ind]");
11: /* message sent to the nearest Leader to A_i */
12: **else**
13: $proposal^{A_i} \leftarrow$ *tuple*;
14: $propState^{A_i} \leftarrow$ true;
15: **for all** $A_j\ in\ children^{A_i}$ **do**
16: var $\leftarrow CommonVar$(A_i, A_j);
17: ind $\leftarrow$ index(Var^{A_i}, var);
18: *sendMsg*(self, A_j,"*ProcessProposal*:$proposal^{A_i}$ *for*:var *at*:ind *withPropState*:$propStateA_i$");
19: **end for**
20: **end if**
21: **end if**
22: **end if**

true for all the states of its children, and its variables are already well instantiated then it will set its state to true and inform the Interface agent of the end of the solving process.

9.4 Experimental comparative evaluation

We have developed the multi-agent dynamic with Actalk [15], an object-oriented, concurrent programming language using the Smalltalk-80 environment. In our experiment, we generated random constraint problems. The parameters used for a meeting problem are: *n* variables in the system, *d* size of the maximal domain, *p* the density of the problem, and *q* the tightness of the constraints.

Our goal in this section is to evaluate the performance of the new DisAS approach, especially on the most hard problems. For this purpose, we conducted two branches of exper-

iment. In the first branch, we generated random problems near the peak of difficulty [24] with n=15, d=5, p=30% and q varied from 25% to 85%. For each $\langle p, q\rangle$ we generated 5 instances. Then we measured the average of the obtained results. These results are expressed in terms of three criteria: the number of constraint checks, the CPU time, and the number of exchanged messages. Note that these are our first experiments.

Table 9.1. Results in mean of constraints checks and CPU time.

	$\langle 0.3, 0.25\rangle$	$\langle 0.3, 0.35\rangle$	$\langle 0.3, 0.45\rangle$	$\langle 0.3, 0.55\rangle$
Constraint Checks	684.8	659.2	542	446
CPU time	112.2	170.4	177.75	198.6
Nber of Messages	112.4	305.2	322	363.4
	$\langle 0.3, 0.65\rangle$	$\langle 0.3, 0.75\rangle$	$\langle 0.3, 0.85\rangle$	
Constraint Checks	387	402.6	379.6	
CPU time	194	241.4	260.6	
Nber of Messages	326.6	398.6	438.6	

We used the same parameters as those given by most researchers when solving distributed complex problems. Table9.1 shows that this approach required a low number of constraint checks and consequently less CPU time and fewer exchanged messages (compared to the results presented in [54, 98], where for example, in [54] the required number of constraint checks for this same parameters is very high, i.e., in most cases, this number varies from 2000 and 10000 ccks.). The notices substantial increasing in the amount of constraint checks, can be justified by the use of the knowledge collected during the lazy enforcement of arc consistency (the set of supports), i.e., many redundant constraint checks are avoided.

For the second group of experiments. We focused our goal on evaluating the performance and efficiency of DisAS vs. AWC search algorithm [110]. This algorithm performs better than ABT due to its dynamic variable ordering, where a bad decision taken by a higher order agent can be easily revised without conducting any exhaustive search.

We randomly generated a set of hard instances according also to the same parameters given by most researchers, n=15; d=5; p=30% and q varying from 0.45 to 0.95 with step of 0.1. We generated about 10 instances for each $\langle p, q\rangle$. We carried our second experiments only on consistent problems. The results reported in Figure9.2 illustrates the obtained outcome in terms of number of constraint checks. We can say that DisAS requires considerably less constraint checks than AWC search to prove the consistency of each instance, e.g., For p=0.3 and q=0.55, AWC search needs seven times the amount of constraint checks performed by DisAS.

Note that for $p \in \{0.25, 0.35\}$, almost all generated problems are inconsistent. In almost all the instances, DisAS detects their inconsistency during the first step.

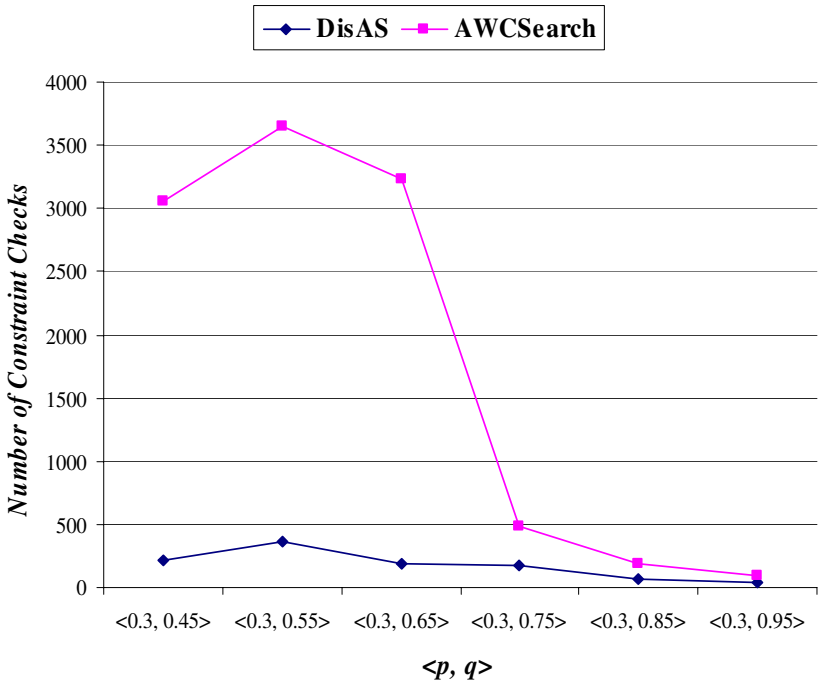


Figure 9.2. DisAS approach vs. AWC Search approach results in mean of the number of constraint checks for binary random CN.

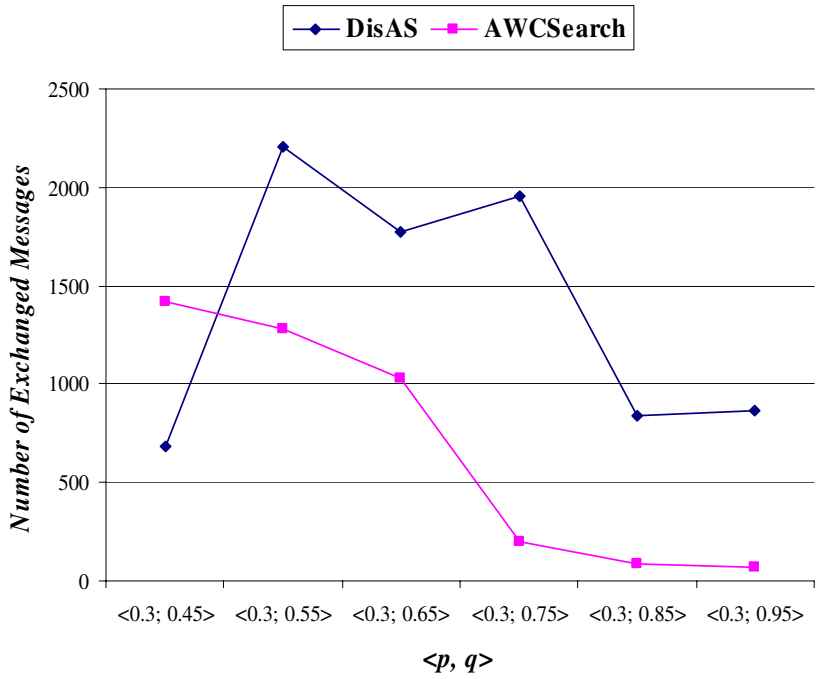


Figure 9.3. DisAS approach vs. AWC Search approach results in mean of the number of exchanged messages for binary random CN.

As for the results expressed in terms of exchanged messages given in Figure9.3. At first glance, it seems that AWC search overtake DisAS. However, it is evident the number of exchanged messages grows with the number of entities in the system. For AWC search the number of agent is only 15 agents, which is the same as the number of variable. While for DisAS, the number of agent is the number of constraints, which is equal to 30% of n^2, $\simeq 30$ agents for each instance. Nevertheless, the use of AWC search for any non-binary problems will require the additions of new agents (new variables), which is not the case for DisAS, also for real-life problems, usually the number of involved constraints in less than the number of variables. This will be as a part of our perspective.

9.5 Summary

We have presented in this chapter a new distributed asynchronous approach to solve any constraints network (DisAS for Distributed Asynchronous Search). The proposed multi-agent model is based on a dual graph representation of CSP in which each agent maintains a constraint of the problem. These agents cooperate concurrently and asynchronously without any central control. However, in addition we proposed a new distributed method to establish a total order among agents with the minimum number of connections. The main reason is to make it easy to use in a real distributed environment and also to decrease the required exchanged messages.

There are two main ideas underlying this approach. First is to perform lazy enforcement of arc consistency, in order to avoid basing high order agents' decisions on arc-inconsistent values. Second is that in case of a conflict, a backjumping is performed to the leader of the concerned shared variable and not to the nearest parent. The reason for this is to avoid all the useless backtracking that can be done between the agent source of the conflict and the first agent responsible of the concerned variable.

In addition, we d not perform either nogood recordings or no new links addition in the new approach. This approach includes an enhanced detection mechanism.

Chapter 10

Conclusions and Future Work

Constraint satisfaction problem (CSP) is a potent formalism to express and to solve many ranges of NP-complete real-world problems, such as planning, resource allocation, time tabling and meeting scheduling. This problem of scheduling meetings (MS) is one the traditional real world problems that continues to fascinate many researchers. This problem embodies a decision-making process affecting several users, in which it is necessary to decide when and where one or more meeting(s) should be scheduled according to several restrictions related to users, meetings, environment, etc. This problem can be naturally expressed using CSP formalism.

A CSP is a triplet (X, D, C) composed of a finite set of *n* variables X, each of which is taking values in an associated finite domain D and a set of *e* constraints C between these variables. Solving a CSP consists in finding one or all-complete assignments of values to variables satisfying all the constraints. This task is hard and many efforts were devoted towards enhancing it by reducing the complexity of the original problem. Hence, this paradigm is marked by the ubiquitous use of local consistency properties and their corresponding enforcement techniques. The basic of these techniques is to prune values that cannot belong to any solution and this in order to reduce the search space and consequently enhance the efficiency of the constraint solver. Many levels of local consistency have been proposed in the literature; among them, reinforcing arc-consistency is the most preeminent one because of its low time and space complexities. Many centralized approaches for reinforcing arc consistency have been proposed in the literature.

However, with the advents of both distributed computing and networking technologies, and due to the natural distribution of many real CSP applications, recently, some efforts were devoted toward centralized techniques. Furthermore, the majority of constraint programming techniques are devoted to binary constraints, i.e., problems where all the constraints imply each at most two variables. Only very few techniques deal directly with n-ary constraints (non-binary constraints). Note that last years the interest to n-ary constraint network (CN) has largely increased, but such algorithms have not been widely studied yet.

10.1 Conclusions

The most important results and contributions from the work presented in this thesis are the following:

- **DRAC and G-DRAC, two new approaches for any CN** We have proposed a new distributed approach for reinforcing arc consistency for binary constraints (that we called DRAC for Distributed Reinforcement of Arc Consistency) based on a Multi-Agent system. This approach has been implemented with Actalk (with Smalltalk-80 environment) and compared with the best existing centralized approach (AC-7) on the basis of randomly generated samples of the phase transition (the most difficult generated problems). The experimental results show that our approach outperforms the existing one in term of constraint checks [14, 15, 16]. Therefore, our second objective was to focus our research on i). Improving DRAC approach by integrating some new heuristics [1], ii) Adapting directly DRAC approach to general constraint network (n-ary constraints). The new approach G-DRAC [6, 1] was implemented and compared to the best existing centralized one (no distributed approach for enforcing AC on n-ary constraints).

- **DRAC^{++}, to deal with more complex problems** We have proposed an improvement of DRAC approach to perform more than arc-consistency. The main motivation is that for some hard constraint network performing only arc-consistency is fruitless because it may not prune any values, or prune only few inconsistent values. Therefore achieving more local consistency pruning levels, with reasonable cost, can be worthwhile. Hence, we should find the best compromise between the cost of the filtering process and the amount of deleted values. Our main contribution is to refine DRAC approach to perform restricted path consistency property (RPC) with the minimum amount of additional constraint checks. The experimental comparative evaluation shows that the new approach, that we called DRAC^{++}, is worthwhile especially for over-constrained problems [10, 11, 8, 4].

- **MSS, a novel static agent-based meeting scheduling solver** We have proposed a novel, complete, deterministic, and static approach to solve any static meeting scheduling problem. For which we proposed two types of constraints, hard constraints, i.e. to express the non-availability of a user that cannot be relaxed, and soft constraints, i.e. to define the preferences of the user that can be relaxed. This discrimination allows us to more closely reflect real applications. In our proposed model, an MS problem is viewed as a set of distributed reactive agents in communications. Each of them acts on behalf of one user. The final result is obtained as a consequence of agents' interactions, each having a local goal. All the agents act in parallel and asynchronously

via sending point-to-point messages. All the agents of the system negotiate by exchanging only necessary relevant information in such a manner to reduce the amount of messages and especially to preserve as much as possible users' privacy. The local goal for each agent is reach the Higher Utility for each scheduled meeting according to defined criteria. The global goal of all the agents is to schedule all the meetings of all the users while satisfying all the inter-agent constraints and achieving the Higher Utility for each scheduled meeting [2, 5, 12].

- **MSRAC, to deal with any meetings' alterations** We have proposed a new approach to deal with any dynamic MS problem. This new approach is expected to be incremental and able to process alterations (the integration of a new meeting and/or cancelation of an already scheduled one). Our main target is to allow the processing of all kind of conflicts among meetings (especially in the case of meetings with same importance). Therefore, to solve this kind of conflicts, we proposed three issues: accept always the meeting with higher local utility, choose randomly one meeting, or apply the metropolis criterion to solve the conflict, i.e., accepting some deterioration in the local utility may increase the global utility [3, 7, 13].

- **DisAS, a new asynchronous solver in the ABT family** Finally we proposed a novel, multi-constraint asynchronous search approach for any constraint network (n-ary constraints). The proposed approach is based in a part on a lazy version of the G-DRAC approach, and without adding any new links and without recording any nogoods as for the existing techniques in the literature. The idea behind using a lazy version of G-DRAC is to save as many as possible fruitless backtracking and consequently to enhance the efficiency of the solving process. We have proposed a new generic distributed method to compute a static constraints ordering were proposed, in which we save as many links as possible leading hopefully to decrease the set of exchanged messages.

10.2 Future work

Our study on ways of solving combinatorial problems and especially meeting scheduling problems stir up our attention to do more further investigations on other challenging research points that have not been address yet. In the following we will discuss some of them.

- **N-ary constraints** As first perspective of our research, we propose to improve DisAS the asynchronous constraint-based solver to deal with any CN, make an exhaustive empirical and theoretical study, and address a practical problem that arises in the real world. We will try to examine the behavior of our proposed work directly on the n-ary problem and also on its encoded binary version.

- **More sophisticated personal computer assistant** Our second perspective is to make our meeting scheduling solver more sophisticated personal computer assistant while integrating the three following issues:

  – **Temporal reasoning**: For this first issue, the events are handled in our system independently. To find a good schedule does not require specification of any relation between two events E_i and E_j. If E_i and E_j have the same welfare, for the participants, to be scheduled at date d_1 and d_2 then choosing randomly one date for each event is enough. Two possible solutions are possible in such situation. However, if the event E_j can be scheduled only when E_i is already planned, then this problem can have only one solution. This temporal relationship among events can be expressed using temporal constraints satisfaction formalism [29]. Khatib et al. proposed in [60] a framework to define soft temporal constraints based on the temporal constraint satisfaction formalism.

    An extension of the proposed formulation (called *temporal constraint satisfaction problems with preferences*) can be used to express temporal aspect among preferences of users that can be relaxed. The main reason is that for MS problems, the preferences of the users are naturally dynamic. In real world the probability of alteration in the calendar of a user differs from one person to another according to his importance in the company. Hence, to formalize dynamic preference of a user we need to use a dynamic temporal constraint reasoning formalism with preferences.

  – **Uncertainty**: For the second issue, the integration of the uncertainty in solving meeting scheduling problem, we noticed that for some cases the arrival of new meetings may lead to a huge perturbation in the already generated schedule. However, in the majority of companies, it is practically impossible to know before hand all the possible coming meetings, and when a sudden meeting occurs it may lead to the reschedule of other meetings or sometimes may lead even to the cancelation of some of them. This may in most cases trouble all the participants. Therefore we need to make some assumptions about future meetings (even a fuzzy view of the coming meetings) to establish some how a stable schedule.

    The idea consists of integrating in the system some uncertain events, depending on the current knowledge about the participant and the status of the company, and trying to include them in the solving process. Luo et al. propose to formulate a static MS problem using Fuzzy constraints [63]. Their idea is to make the system more flexible by integrating soft constraints which can be partially violated and this by using fuzzy constraints to express only users' preferences and not to represent possible coming meetings. In a different manner we will formalize the MS problem using an extended version of the CSP formalism able to represent real-

life scenario especially where the knowledge is not completely available. One possible idea is to use in our modeling SCSP [18] (for semiring-based CSP) formalism to express the level of consistency of the constraints (probability that the event will occur). However, a semiring is associated to the standard definition of a CSP, so that different choices of the semiring represent different concrete constraint satisfaction schemes. In addition we will try to make our system more flexible in the sense that a meeting may be planned even if some participants are not present. This issue depends in a part on the participants, the head of the meeting and also on the number of absent participants.

– **Learning process**: For the third issue, one of the fundamental aspects of a personal computer assistant is that it is enduring and self-improving. It is expected to persist indefinitely and learn over time to make good decisions that better reflect user constraints and preferences. Therefore, we will try to address the problem of how to make the autonomous agents of the system, that sense and act on behalf of the users, learn from the already processed meetings and thus they become able to choose optimal actions to achieve their goals. Let's recall that these agents should cooperate and coordinate in the sense to jointly reach a consensus over which actions to perform. For that, we will try to integrate learning process and consequently to decrease the cost of the communication and increase the performance of the system. Agents must learn to coordinate their actions through other agents' feedbacks. This learning process requires the access to the user's calendar, the incoming and outgoing meeting requests, the answer of the user and especially the confirmed time slots.

  However, integrating learning process in our approach may lead to an increase in the privacy loss because; learning process requires recording some knowledge about other agents (other human users) to avoid repeating the same task (asking for the same place or asking same person twice about same date already rejected). However, in many settings, users may want to maintain their privacy as much as possible while still engaging in a collaborative task. Only in recent years that this concern has been taken into consideration while designing new techniques for solving real-world problems and evaluating them. Even with limited communication policy, an agent (acting on behalf of a human user) may collect and deduce some private information about other agents. Hence, to have efficient system raises the question of how to integrate learning process while maintaining as much as possible the privacy of the implied users. Therefore, we need to find good compromise between the revealed information and the gain obtained from the learning process. Thus, the main problem is how to reach agreement among all the agents of the system without revealing private information. In addition, the calendars of the participants are dynamic, and a rejected date by a participant

may be accepted later. Maintaining the consistency of the users' calendars may affect the efficiency of the learning process.

Bibliography

[1] Abdennadher, S. and Schlenker, H., Nurse scheduling using constraint logic programming. *In Proceedings Eleventh Annual Conference on Innovative Applications of Artificial Intelligence, IAAI*-99, pp. 838-843, 1999.

[2] Abrahamson, K., Dadoun, N., Kirkpatrick, D. G. and Przytycka, T., A Simple Parallel Tree Contraction Algorithm. *In Journal Algorithms*, Vol. 10, No.2, pp. 287-302, 1989.

[3] Baudot, B., Deville, Y., Analysis of Distributed Arc Consistency Algorithms. *Technical Report, RR*-97-07, 1997.

[4] Bakker, R.R., Dikker, F., Tempelman, F., Wognum P. M., Diagnosing and Solving Over-determined Constraint Satisfaction Problems. *In Proceedings of IJCAI*-93, pp. 276-281, 1993.

[5] Berlandier, P., Improving Domain Filtering Using Restricted Path Consistency. *In Proceedings of IEEE CAIA*-95, pp. 32-37, 1995.

[6] Bessière, C., Arc consistency and arc consistency again. *In Artificial Intelligence Journal*, Vol. 65, pp. 179-190, 1994.

[7] Bessière, C., Freuder, E.C., Regin, J-C., Using Inference to Reduce Arc Consistency Computation. *In Proceedings of IJCAI*-95, pp. 592-598, 1995.

[8] Bessière, C., Freuder, E., and Régin, J-C., Using constraint Metaknowledge to Reduce Arc Consistency Computation. *In Artificial Intelligence Journal*, Vol. 107, pp. 125-148, 1999.

[9] Bessière, C. Régin, J-C., Refining the Basic Constraint Propagation Algorithm. *In Proceedings of IJCAI*-01, pp. 309-315, 2001.

[10] Bessière, C., Hentenryck, P.V., To Be or Not to Be ... a Global Constraint. *In the proceedings of CP*-03, pp. 789-794, 2003.

[11] Bessière, C., Regin, J-C., Arc consistency for general constraint networks : preliminary results. *In Proceedings of IJCAI*-97, pp. 398-404, 1997.

[12] Bessière, C., Meseguer, P., Freuder, E.C., and Larossa J., On Forward Checking for non-binary Constraint Satiafaction. *In Proceedings of CP*-99, pp. 88-102, 1999.

[13] Bessière, C., Brito, I., Maestre, A., and Meseguer, P., Asynchronous Backtracking without Adding Links: A New Member in the ABT Family. *In Artificial Intelligence Journal*, vol. 161, pp. 7-24, 2005.

[14] Bessière, C., Regin, J-C., Enforcing Arc Consistency on Global Constraints by Solving Subproblem on the Fly, *In Proceedings of CP*-99, pp. 103-117, 1999.

[15] Briot, J.P., Actalk., A Framework for Object-Oriented Concurrent Programming - Design and Experience. *In Object-Oriented Parallel and Distributed Programming, Hermes Science Publications*, pp. 209-231, 2000.

[16] Bacchus, F., Chen, X., Beek, P.V., and Walsh T., Binary vs. non-binary constraints. *In the Artificial Intelligence Journal*, Vol. 140, No. 1-2, pp. 1-37, 2002.

[17] Bliek, C., and Sam-Haroud, D., Path Consistency on triangular Constraint Graphs. *In the Proceedings of IJCAI*-99, pp. 456-461, 1999.

[18] Bistarelli, S. Montanari, U., and Rossi F., Constraint Solving over Semirings. *In Proceedings of IJCAI*-95, pp. 624-630, 1995.

[19] Chandy, K.M. and Lamport, L., Distributed snapshots: Determining global states of distributed systems. *ACM Trans. on Comp. Systems*, Vol. 3, No. 1, pp. 63-75, 1985.

[20] Chmeiss, A. and Jegou, P., New Constraint Propagation Algorithms Requiring Small Space Complexity. *In Proceedings of the 8^{th} IEEE ICTAI*-96, pp. 286-289, 1996.

[21] Cooper, P. R., Swain, M.J., Arc consistency: Parallelism and Domain Dependence. *In Artificial Intelligence Journal*, Vol. 65, pp. 179-190, 1994.

[22] Conry S.E., Kuwabara K., Lesser V.R., and Meyer R.A. Multistage Negotiation for Distributed Constraint Satisfaction. *In IEEE Transactions on Systems, Man and Cybernetics*, Vol. 21, No.6, pp. 1462-1477, 1991.

[23] Chmeiss, A. and Jegou, P., New Constraint Propagation Algorithms Requiring Small Space Complexity. *In Proceedings of the 8th IEEE ICTAI*-96, pp. 286-289, 1996.

[24] Cheesman, P., Kanefsky, B., and Taylor, W., Where the Really Hard Problems are. *In Proceedings of the 12^{th} IJCAI*-91, pp. 331-337, 1991.

[25] Dent, M.J., Mercer, R.E., Minimal Forward Checking. *In Proceedings of the 6th ICTAI*-94, pp. 432-438, 1994.

[26] Dechter, R. and Dechter, A., Belief maintenance in dynamic constraint networks. *In Proceedings 7th National Conference on Artificial Intelligence, AAAI*-88, pp.37-42, 1988.

[27] Dechter, R., Pearl, J., Tree-Clustering Schemes for Constraint-Processing. *In Proceedings of AAAI*-88, pp. 150-154, 1988.

[28] Dechter, R., On the Expressiveness of Networks with Hidden Variables. *In the Proceedings of the Eight National Conference on Artificial Intelligence*, pp. 556-562, 1990.

[29] Dechter, R., Meiri, I and Pearl, J., Temporal constraint networks. *In Artificial Intelligence Journal*, Vol. 49, pp. 61-95, 1991.

[30] Deville, Y. and Hentenryck, P. V., En efficient arc consistency algorithm for a class of CSP problems. *In Proceedings of the 12th IJCAI*-91, pp. 325-330, 1991.

[31] Debruyne, R. and Bessière, C., Some Practical Filtering Techniques for Constraint Satisfaction Problems. *In Proceedings of IJCAI*-97, pp. 412-417, 1997.

[32] Debruyne, R. and Bessière, C., From Restricted Path Conssitency to Max Restricted Path Consistency. *In Proceedings CP*-97, pp. 312-326, 1997.

[33] Debruyne, R., and Bessière, C., Domain fitering consistencies. *In Journal of Artificial Intelligence Research*, Vol. 14, pp. 205-230, 2001.

[34] Ephrati, E. and Rosenschein, J. S., The Clarke Tax as a Consensus Mechanism Among Automated Agents. *In Proceedings of the 9th National Conference on Artificial Intelligence*, pp. 173-178, 1991.

[35] Ephrati, E., Zlotkin, G. and Rosenschein, J. S., A Non-manipulable Meeting Scheduling System. *In Proceedings International Workshop on Distributed Artificial Intelligence*, pp. 105-125, 1994.

[36] Freuder, E.C., Wallace, R.J., Partial Constraint Satisfaction. *In Artificial Intelligence Journal*, Vol. 58, No. 1-3, pp. 21-70, 1990.

[37] Freuder, E.C., A Sufficient Condition for Backtrack-free Search. *In Journal of the ACM*, Vol. 29, No. 1, pp. 24-32, 1982.

[38] Freuder, E.C., A Sufficient Condition for Backtrack-bounded Search. *In Journal of the ACM*, Vol. 32, No. 4, pp. 755-761, 1985.

[39] Freuder, E., and Elfe, C., Neighborhood Inverse Cnsistency Preprocessing. *In the Proceedings of AAAI*-96, pp. 202-208, 1996.

[40] Feldman, A.M. *Welfare Economics and Social Choice Theory*. Kluwer, Boston, 1980.

[41] Franzin, M.S., Freuder, E.C., Rossi, F. and Wallace, R., Multi-agent meeting scheduling with preferences: efficiency, privacy loss, and solution quality. *In Proceedings of AAAI-2002 workshop on preference in AI and CP*, 2002.

[42] Franzin, M.S., Freuder, E.C., Rossi, F. and Wallace, R., Multi-agent meeting scheduling with preferences: efficiency, Solution quality and privacy loss. *In the special issue of computational Intelligence on Preferences in AI and CP*, Vol. 20, No. 2, pp. 264-286, 2004.

[43] Garey, M. R., and Johnson, D. S., *Computers and Intractability- A guide to the Theory of NP-Completeness*. W.H.Freeman&Co, 1979.

[44] Grandoni, F., and Italiano, G. F., Improved algorithms for max-restricted path consistency. *In Proceedings of Principles and Practice of Constraint Programming CP*-03, pp. 858-862, 2003.

[45] Golomb, S.W., and Baumert, L.D., Backtrack Programming. *Journal of the ACM*, Vol.12, No.4, pp. 516-524, 1965.

[46] Gaschnig, J., Performance Measurement and Analysis of Certain Search Algorithms. *Technical Report CMU-CS-79-124, Carnegie-Mellon University, Pittsburgh, PA*, 1979.

[47] Gaschnig, J., A General Backtrack Algorithm that Eliminates most Redundant Tests. *In Proceedings of the IJCAI*-77, Vol. 1, pp. 457, 1977.

[48] Grant, S.A. and Smith, B.M., The phase transition behavior of maintaning arc-consistency. *In Proceedings of ECAI-96*, pp. 175-179, 1996.

[49] Garrido, L. and Sycara, K. Multi-Agent Meeting Scheduling: Preliminary Experimental Results. *In Proceedings 2nd International Conference Multi-Agent Systems, ICMAS*-96, pp. 95-102, 1996.

[50] Getoor, L., Ottosson, G., Fromherz, M., and Carlson, B., Effective Redundant Cosntraints for Online Scheduling. *In Proceedings of fourtheenth National Conference on Artificial Intelligence*, pp. 302-307, 1997.

[51] Goldwasser, S., and Bellare, M., Lecture notes on cryptography. *MIT Press*, 1996.

[52] Gent, I., Underwood, J., The Logic of Search Algorithms: Theory and Applications. *In Proceedings of the Third International Conference on Principles and Practice of Constraint Programming, CP*-97, pp. 77-91, 1997.

[53] Hamadi, Y., Optimal Distributed Arc-Consistency. *In Proceedings of Constraint Programming*, pp. 219-233, 1999.

[54] Hamadi, Y., Bessière, C., and Quinqueton, J., Backtracking in Distributed Constraint Networks. *In proceedings of ECAI*-98, pp. 219-223, 1998.

[55] Huhns M.N., and Bridgeland D. M., Multiagent Truth Maintenance. *In IEEE Transactions on Systems, Man and Cybernetics*, Vol. 21, No. 6, pp. 1437-1445, 1991.

[56] Haralick, R., Elliot, G., Increasing Tree Efficiency for Constraint Satisfaction Problems. *In Artificial Intelligence Journal*, Vol. 14, pp. 263-314, 1980.

[57] Hentenryck, P. Van, Constraint Satisfaction in Logic Programming. *MIT Press*, 1989.

[58] Hentenryck, P. Van., Deville, Y. and Teng, C-M. A generic arc-consistency algorithm and its specifications. *In Journal Artificial Intelligence Research*, Vol. 27, pp. 291-322, 1992.

[59] Kondrak, G., Van Beek, P., A Theoretical Evaluation of Selected Backtracking Algorithms. *In Aritificial Intelligence Journal*, Vol. 89, pp. 365-387, 1997.

[60] Khatib,L., Morris, P., Morris, R., and Rossi F., Temporal constraint reasoning with preferences. *In Proceedings of IJCAI*-01, pp. 322-327, 2001.

[61] Lee, J., and Van Emden, M., Interval Computation as Deduction in CHIP. *In Journal of Logic Programming*, Vol. 16, pp. 255-276, 1993.

[62] Lemaitre, M. and Verfaillie, G., An Incomplete Method for Solving Distributed Valued Constraint Satisfaction Problems. *In Proceedings AAAI-97 Workshop on Constraints and Agents*, 1997.

[63] Luo, X., Leung,H. and Lee, J.H., Theory and Properties of Selfish Protocol for Multi-Agent Meeting Scheduling Using Fuzzy Constraints. *In Proceedings of* 14^{th} *ECAI*-00, pp.373-377, 2000.

[64] McGregor, J.J., Relational consistency algorithms and their application in finding subgraph and graph isomorphism. *In Information Science*, Vol. 19, pp. 229-250, 1979.

[65] Mackworth, A. K., Consistency in Networks of Relations. *In Artificial Intelligence Journal*, Vol. 8, pp. 99-118, 1977.

[66] Mackworth, A. K., On reading sketch maps. *In Proceedings IJCAI*-77, pp. 598-606, 1977.

[67] Montanari, U., NetWorks of Constraints: Fundamental Properties and Applications to Picture Processing. *In Information Sciences*, Vol. 7, pp. 95-132, 1974.

[68] Maheswaran, R.T., Tambe, M., and Bowring E., Taking DCOP to the Real World: Efficiet Complete Solutions for Distributed Multi-Event Scheduling. *In the Proceedings of AAMAS*-2004, pp. 310-318, 2004.

[69] Maheswaran, R.T., Pearce J.P., varakantham P., Bowring E., and Tambe M., Valuation of possible states (VPS):A Quantitative Framework for Analysis of Privacy Loss Among Collaborative Personal Assistant Agents. *In the Proceedings of AMAS*-2005, (to appear).

[70] Maestre, A., and Bessière, C., Improving Asynchronous Backtracking for Dealing with Complex Local Problems. *In Proceedings of ECAI*-04, pp. 206-210, 2004.

[71] Mohr, R. and Henderson, T. C., Arc and path consistency revisited. *In Artificial Intelligence Journal*, Vol. 28, pp. 225-233, 1986.

[72] Mohr, R., Masini, G., Good old discrete relaxation. *In Proceedings ECAI*-88, pp. 651-656, 1988.

[73] Minton, S., Johnston, M.D., Philips, A.B., and Laird P., Minimizing Conflicts: A Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems. *In Artificial Intelligence Journal*, Vol.5 8, No. 1-3, pp. 161-205, 1992.

[74] Morris, P., The Breakout Method for Escaping from Local Minima. *In Proceedings of the 11^{th} National Conference on Artificial Intelligence*, pp. 40-45, 1993.

[75] Nash, J. F., Two-person Cooperatives Games. *In Econometrica*, Vol. 21, pp. 128-140, 1953.

[76] Nadel, B.A., Representation Selection for Constraint Satisfaction: A Case Study Using *n*-queens. *In the IEEE Expert*, Vol. 5, pp. 16-23, 1990.

[77] Nguyen, T. and Deville, Y., A Distributed Arc-Consistency Algorithm. *In the First International Workshop on concurrent Constraint Satisfaction*, 1995.

[78] Nguyen, T. and Deville, Y., A Distributed Arc-Consistency Algorithm. *In Science of Computer Programming*, Vol. 30, pp. 227-250, 1997.

[79] Perlin

[80] Prosser, P., Hybrid Algorithms for the Constraint Satisfaction Problem. *In Computational Intelligence*, Vol. 9, No.3, pp. 268-299, 1993.

[81] Prosser, P., Forward Checking with Backmarking. *In Constraint Processing* (LNCS), Vol. 923, pp. 185-204, 1995.

[82] Puget, J.-F., A Fast Algorithm for the Bound Consistency of Alldiff Constraints. *In Proceedings of AAAI*-98, pp. 359-366, 1998.

[83] Rao., V.N. and Kumar, V., On the Efficiency of Parallel Backtracking. *In the IEEE transactions on Parallel and Distributed Systems*, Vol.4, No. 4, pp. 427-437, 1993.

[84] Regin, J. C., A filtering algorithm for constraints of difference in CSPs. *In Proceedings AAAI*-94, pp. 362-367, 1994.

[85] Mullender, S., *Distributed Systems*. Addison-Wesley, Second Edition, 1995.

[86] Sabin, D., and Freuder, E.C., Contradicting Conventional Wisdom in Constraint Satisfaction. I*n Proceedings ECAI*-94, pp. 125-129, 1994.

[87] Samal, S., Henderson, T.C., Parallel Consistent Labeling Algorithms. *In International Journal of Parallel Programming Archive*, Vol. 16, pp. 341-364, 1987.

[88] Schiex, T, Fargier, H. and Verfaillie, G., Valued Constraint Satisfaction Problems: Hard and Easy Problems. *In Proceedings of 14th IJCAI*-95, pp. 631-637, 1995.

[89] Singh, M.: Path Consistency revisited. *In Proceedings of the 7th IEEE ICTAI*-95, pp. 318-325, 1995.

[90] Stergiou, k., Walsh, T., The Difference All-Difference Makes. *In Proceedings of IJCAI*-99, pp. 414-419, 1999.

[91] Stergiou, K., Representation and Reasoning with non-binary Constraints. *PhD dissertation, Departement of Computer Science, Glasgow, Scotland,* 2001.

[92] Sen, S., Haynes, T., Arora, N., Satisfying User Preferences While Negotiating Meetings. In Inter. *In Journal of Human-Computer Studies*, Vol. 47, pp. 407-427, 1997.

[93] Sen, S. and Durfee, E.H. On the design of an adaptive meeting scheduler. *In Proceedings of 10^{th} IEEE Conference AI Applications*, pp. 40-46, 1994.

[94] Smith, B., Brailsford, S.C., Hubbard, P.M., and Williams, H.P., The Progressive Party Problem: Integer Linear Programming and Constraint Programming Compared. *In Constraints*, Vol. 1, pp. 119-138, 1996.

[95] Sycara K.P., Roth S., Sadeh N., and Fox M.S. Distributed Constrained Heuristic Search. *In IEEE Transactions on Systems, Man and Cybernetics*, Vol. 21, No. 6, pp. 1446-1461, 1991.

[96] Sycara, K. and Liu, J.S. Distributed meeting scheduling. *In Proceedings 16th Annual Conference of Cognitive Society*, 1994.

[97] K. N. Sivarajan, R. J. McEliece, J. W. Ketchum., Channel Assignment in Cellular Radio, *In Proceedings of the 39th IEEE Vehicular Technology Conference*, pp. 846-850, 1989.

[98] Silaghi, M.-C., Sam-Haroud D., and Faltings, B.V. Asynchronous Search with Aggregations. *In Proceedings of the 17th National Conference on Artificial Intelligence AAAI*-00, pp. 917-922, 2000.

[99] Schwald, E., and Dechter, R., Processing Disjunctions in Temporal Constraint Networks. *In Artificial Intelligence Journal*, Vol. 93, pp. 29-61, 1997.

[100] Tsang, E., Foundations of Constraint Satisfaction. *In Computation in Cognitive Science, Academic Press*, 1993.

[101] Tsuruta, T. ans Shintani, T. Scheduling Meetings using Distributed Valued Constraint Satisfaction Algorithm. *In Proceedings 14th ECAI*-00, pp. 383-387, 2000.

[102] Van Beek, P., Dechter, R. On the minimality and global consistency of row-convex con-straint networks. *Journal of the ACM*, Vol. 42, No. 3, pp. 543-561, 1995.

[103] Verfaillie, G., Martinez, D., and Bessiere, C., A Generic Customizable Framework for Inverse Local Consistency. *In Proceedings of AAAI*-99, pp. 169-174, 1999.

[104] Waltz, D. L., Understanding Line Drawings of Scenes with Shadows. *In the Psychology of Computer Vision, McGraw Hill*, pp. 19-91, 1975 (first published in:Tech.Rep. AI271, MIT MA, 1972).

[105] Wooldridge, M., *An Introduction to Multi-Agent Systems*. John Wiley & Sons, LTD, 2002.

[106] Yokoo, M. Ishida. T, and Kuwabara, K. Distributed Constraints Satisfaction for DAI Problems. *In 10th International Workshop in Distributed Artificial Intelligence (DAI-90)*, 1990.

[107] Yokoo, M. and Hirayama, K. Algorithms for Distributed Constraints Satisfaction: A Review. *In International Journal Autonomous Agent and Multi-Agent Systems*, Vol. 3, No. 2, pp. 185-207, 2000.

[108] Yokoo, M., Suzuki, K., and Hirayama, K., Secure distributed constraint satisfaction: Reaching agreement without revealing private information. *In Principles and Practice of Constraint Programming, CP*-02, LNCS 2470, pp.387-401, 2002.

[109] Yokoo, M., Durfee, E.H., Ishida, T., and Kuwabara, K., The Distributed Constraint Satisfaction Problem: formalism and algorithms. *In the IEEE Transactions on Knowledge and Data Engineering*, Vol.10, No. 5, pp. 673-685, 1998.

[110] Yokoo, M., Asynchronous Weak-commitment Search for Solving Distributed Constraint Satisfaction Problems. *In the Proceedings of the First International Conference on Principles and Practice of Constraint Programming, CP*-95, pp.88-102, 1995.

[111] Yokoo M., Durfee E.H., Ishida T., and Kuwabara K. Distributed Constraint Satisfaction Formalizing Distributed Problem Solving. *In Proceedings of DCS*, pp. 614-621, 1992.

Publications

International Journals

1. Ben Hassine, A. and Ho, T.B., ”How to Enforce Local Consistency on any Constraint Network by Reactive Agents”. Submitted to the International Journal of Artificial Intelligence Tools (IJAIT), 30 pages, World Scientific Published (Under Review) 2004.

2. Ben Hassine, A., Ito, T. and Ho, T. B., ”Meetings Scheduling Solver Enhancement with Local Consistency Reinforcement”. The International Journal of Applied Intelligence (Kluwer Publishers), 11 pages, 2004 (*to appear*).

3. Ben Hassine, A. and Ho, T. B., ”An Agent-Based Approach to Solve Dynamic Meeting Scheduling Problems with Preferences”. Submitted to the International Journal of Engineering Applications of Artificial Intelligence (EAAI), 30 pages, Elsevier Publisher (Under Review), 2005.

Refereed International Conferences

4. Ben Hassine, A. and Ghédira, K. ”A Distributed Arc-Consistency Reinforcement for Constraint Satisfaction Problems”. In the Acts of the Second Scientific workshop of the Young Researchers in Electronic and Data-processing Engineering, GEI’2002, Hammamet, Tunisia, 2002.

5. Ben Hassine, A. and Ghédira, K. ”How to Establish Arc-Consistency by Reactive Agents”. In Proceedings of the 15th European Conference on Artificial Intelligence, p. 156-160, ECAI-2002 Lyon, French, 2002.

6. Ben Hassine, A. and Ghédira, K. ”Using Reactive Agents to Establish Arc-Consistency”. In Proceedings of the Seventh Pacific Rim International Conference on Artificial Intelligence, p. 97-107, PRICAI-2002 Tokyo-Japan, 2002.

7. Ben Hassine, A., Ito, T. and Ho, T. B. ”A New Distributed Approach to Solve Meeting Scheduling Problems”. In Proceedings IEEE/WIC Int. Conf. Intelligent Agent Technology (IAT’03), Halifax, Canada, 2003.

8. Ben Hassine, A., Xavier Défago and Ho, T. B. ”Nouvelle Approche pour la Résolution Dynamique des Problèmes d’Ordonnancement des Rencontres”. Presented in the Scientific French-speaking Workshops (JSF’03), 24-26 November, Tokyo, 2003.

9. Ben Hassine, A., Ito, T. and Ho, T. B. ”Scheduling Meetings with Distributed Local Consistency Reinforcement”. In the Proceedings of the 17th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE 2004), pp. 679-688, Ottawa, Canada, 2004 (*nominated for the best paper*).

10. Ben Hassine, A., Ghédira, K. and Ho, T. B. ”New Distributed Filtering-Consistency Approach to General Networks”. In the Proceedings of the 17th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE 2004), pp. 708-717, Ottawa, Canada, 2004.

11. Ben Hassine, A., Défago, X. and Ho, T. B. ”Agent-Based Approach to Dynamic Meeting Scheduling Problems”. *In the Third International Joint Conference on Autonomous Agents and Multi Agent Systems* (AAMAS04), pp. 1132-1139, NewYork, 2004, (acceptance rate 24

12. Ben Hassine, A. and Ho, T. B. ”DRAC++ for Distributed Restricted Path Consistency”. In the Proceedings of the Japan-Tunisia Workshop on Computer Systems and Information Technology (JT-CSIT04), Tokyo, 2004.

13. Idrissi, A. and Ben Hassine, A. ”Circuit Consistencies”. *In the Proceedings of 8th Pacific Rim International Conference on Artificial Intelligence*, pp. 124-133, 2004 (acceptance rate 26%).

14. Ben Hassine, A. and Ho, T. B. ”Nouvelle Approche Générique pour le Renforcement Distribué de la Consistance de Chemin Restreint”. In Proceedings of the Scientific French-speaking Workshops (JSF’04), 4-5 November, Tokyo, 2004.

15. Ben Hassine, A. and Ho, T. B. ”Restricted Path Consistency Enforcement for any Constraint Network”. In Proceedings of the Joint Workshop of Vietnamese Society of AI, SIGKBS-JSAI, ICS-IPSJ and IEICE-SIGAI on Active Mining AM’04, (IEICE Technical Report Vol.104 No.485), 4-7 December, Hanoi-Vietnam, 2004.

16. Ben Hassine, A. and Ho, T. B., Performing more than AC for Hard Distributed Constraint Satisfaction Problems. *In the Proceedings of AAMAS-RRS Workshop*, Netherland, 2005 (*to appear*).