Algorithms for irreducible infeasible subset detection in CSP - Application to frequency planning and graph k-coloring

Jun Hu

SPIM

PhD Dissertation

école doctorale **sciences pour l'ingénieur et microtechniques**

UNIVERSITÉ DE TECHNOLOGIE BELFORT-MONTBÉLIARD

Algorithms for Irreducible Infeasible Subset detection in CSP - Application to frequency planning and graph *k*-coloring

HU Jun

27 November 2012

Reviewer	**Steve HURLEY**, Professor University of Wales Cardiff
Reviewer	**Christian ARTIGUES**, Research Director CNRS-LAAS, Université de Toulouse
Examiner (President of Jury)	**Jin-Kao HAO**, Professor Université d'Angers
Examiner	**Hakim MABED**, Maître de Conférences Université de Franche-Comté
Examiner	**Thierry DEFAIX**, PhD - Project manager Direction Générale des Armées - Bruz
Supervisor	**Alexandre CAMINADA**, Professor Université de Technologie de Belfort-Montbéliard
Co-Supervisor	**Philippe GALINIER**, Professor Ecole Polytechnique de Montréal

To my mother

To my father

Acknowledgments

This work is a fruit of my cordial relationship with Alexandre CAMINADA as my advisor. His perpetual encouragement and motivation have illuminated my graduate life. The immense time and the expertise he dedicated make this research possible. His great personality and excellent professional attitude will always be an example for me to follow. I am grateful to Professor Philippe GALINIER as my co-advisor, for his inspired discussions and invaluable comments on my research.

I am deeply indebted for patience and generosity in the transmission of knowledge with Laurent MOALIC, Alexandre GONDRAN, Hakim MABED and Mohammad DIB. I would give a special thank to Dr. Jean-Noël MATIN, who enlightens me with the beauty of mathematics, for his patience and dedication to research.

My thanks goes also to the members in IRTES laboratory: Dr. Yan FEI, HuiDe ZHOU, You LI, Julien MOREAU, Dr. Frédéric LASSABE and Stefan TEODORESCU, with whom, my journey become enjoyable.

I am immensely indebted to my parents for their unconditional and constant love, it would be impossible for me to express my gratitude towards them in mere words. I dedicate this thesis to them.

Résumé

L'affectation de fréquences (AFP) consiste à attribuer des fréquences radio aux liens de communications d'un réseau en respectant un spectre de fréquences donné et des contraintes d'interférence électromagnétique sur les liens. Vu la limitation des ressources spectrales pour chaque application, les ressources en fréquences sont souvent insuffisantes pour déployer un réseau sans interférence. Dans ce cas, le réseau est surcontraint et le problème est irréalisable. Résoudre le problème consiste alors à identifier les zones surcontraintes pour en revoir la conception.

Le travail que nous présentons concerne la recherche d'une de ces zones surcontraintes avec une approche algorithmique basée sur la modélisation du problème par un CSP. Le problème de l'affectation de fréquences doit donc être modélisé comme un problème de satisfaction de contraintes (CSP) qui est représenté par un triplé : un ensemble de variables (les liens radio), un ensemble de contraintes (les interférences électromagnétiques), et un ensemble de domaines (les fréquences admises).

Sous forme de CSP, une zone perturbée peut être considérée comme un sous-ensemble irréalisable irréductible du problème (IIS pour Irreductible Infeasible Subset). Un IIS est un sous problème de taille minimale qui est irréalisable, c'est-à-dire que tous les sous-ensembles d'un IIS sont réalisables. L'identification d'un IIS dans un CSP se rapporte à deux résultats généraux intéressants. Premièrement, en localisant un IIS on peut plus facilement prouver l'irréalisabilité d'un problème donné car l'irréalisabilité d'un IIS, qui est supposé être petit par rapport au problème complet, est plus rapidement calculable que sur le problème entier. Deuxièmement, on peut localiser la raison de l'irréalisabilité; dans ce cas, sur un problème réel, le décideur peut proposer des solutions pour relâcher des contraintes de l'IIS, et peut-être aboutir à une solution réalisable pour son problème. La recherche d'IIS consiste donc à résoudre un problème fondamental qui fait partie des outils de prise de décision.

Ce travail propose des algorithmes pour identifier un IIS dans un CSP incohérent. Ces algorithmes ont été testés sur des instances connues du problème de l'affectation des fréquences et du problème de k-coloration de graphe. Les résultats ont montrés d'une grande amélioration sur des instances du problème de l'affectation des fréquences par rapport aux méthodes connues.

Abstract

The frequency assignment (FAP) consists in assigning the frequency on the radio links of a network which satisfies the electromagnetic interference among the links. Given the limited spectrum resources for each application, the frequency resources are often insufficient to deploy a wireless network without interference. In this case, the network is over-contrained and the problem is infeasible. Our objective is to identify an area with heavy interference.

The work presented here concerns the detection for one of these areas with an algorithmic approach based on modeling the problem by CSP. The problem of frequency assignment can be modeled as a constraint satisfaction problem (CSP) which is represented by a triple: a set of variables (radio links), a set of constraints (electromagnetic interference) and a set of available frequencies.

The interfered area in CSP can be considered a subset of irreducible feasible subset (IIS). An IIS is a infeasible subproblem with irreducible size, that is to say that all subsets of an IIS are feasible. The identification of an IIS in a CSP refers to two general interests. First, locating an IIS can easily prove the infeasibility of the problem. Because the size of IIS is assumed to be smaller compared to the entire problem, its infeasibility is relatively easier to prove. Second, we can locate the reason of infeasibility, in this case, the decision maker can provide the solutions to relax the constraints inside IIS, which perhaps leads to a feasible solution to the problem.

This work proposes algorithms to identify an IIS in the over-constrained CSP. These algorithms have tested on the well known benchmarks of the FAP and of the problem of graph k-coloring. The results show a significant improvement on instances of FAP compared to known methods.

Contents

List of Figures

List of Tables

List of publications

Publications of International conference

- Jun HU, Philippe Galinier and Alexandre Caminada, Yet another breakout inspired infeasible subset detection in constraint satisfaction problem, ICAI 2011, Las Vegas USA, July 18-21 2011.
- Jun HU, Philippe Galinier and Alexandre Caminada, On Identifying Infeasible Subsets in Constraint Satisfaction Problems, ICAI 2010 page 615-619, Las Vegas USA, July 12-15 2010.
- Jun HU, Philippe GALINIER and Alexandre CAMINADA, Find perturbation zone in telecommunication network, INFORMS Telecom 2010, Montreal Canada, May 5-7 2010.
- Jun HU, Alexandre CAMINADA and Hakim MABED, Simple value ordering heuristic in Frequency Assignment Problem, The 39th International Conference on Computers Industrial Engineering, Troyes France, July 6-8 2009.

Publications of National conference

- Jun HU and Alexandre CAMINADA, Résolution itérative du Max-CSP, ROADEF 2010, Toulouse France, February 24-26 2010.
- Jun HU, Alexandre CAMINADA, Hakim MABED, Recherche de zone de blocage dans un graphe. ROADEF 2009, Nancy France, February 10-12 2009.
- Mohammad DIB, Alexandre CAMINADA, Hakim MABED, Jun HU, Propagation de contraintes et gestion de Nogood pour le CSP, ROADEF 2008, Clermond-Ferrand France, February 25-27 2008.

Seminars

- Jun HU, Une méthode hybride pour la recherche de zone de blocage dans le réseau télécommunication, Project meeting with DGA/Silicom, Rennes France, 24 April 2008.
- Jun HU, Méthodes exactes et approchées pour la recherche de zone de blocage dans le réseau télécommunication, Project meeting with DGA/Silicom, Belfort France, 29 November 2007

Introduction

In the human ecosystem, all the resources are limited. For example, the scale of the landscape for living, the resource of drinkable water, etc. A crucial reality is that human beings always attempt to benefit themselves maximally without respecting the natural laws. Such conflict between the human's "greed" and nature's "impracticability" leads to the effect that many objectives cannot be achieved without increasing the indispensable resources, or reducing several implied constraints and requests.

Let's begin with a simple example in the real world which demonstrates the above conflict. Suppose that two people named Tom and Jerry decide to open a bar. Tom is available during the week except Monday and Sunday, Jerry is available during the week except Saturday and Sunday which represent the availability of human resources. Their objective is to maintain the bar open during all days in a week except Monday. It is quite clear to see that such an objective cannot be achieved without compromise, since both Tom and Jerry are not available on Sunday. Such human resource shortage leads to the failure of promise. Imagine another academic example, a graph of a triangle consists of three nodes and three arcs with 2 colors, the graph cannot be colored by assigning two adjacent nodes differently.

From these two very simple examples, it is easy to verify the unsatisfiability of the problems and to find the conflicts. In more complex cases, the unsatisfiability and the conflicts are not so visible. For example, a telecommunication operator wants to assign available frequencies to the antenna in a city telecommunication network while satisfying interference between adjacent antennae. Due to the network scale and the complexity of interference defined by the physical laws related to radio communication, it is very difficult for a human to determine whether the given frequencies is sufficient to avoid interference in the network without any computer aids.

In order to adopt the computer aids, the real world problems need to be described firstly in the form of mathematical models. One convenient modeling technique is to define the real world problem as a constraint satisfaction problem. Such a technique elegantly represents the problems by three basic components – the decision variables, the available resources for each variable, and the constraints to satisfy while assigning resources to the variables.

Considering the above bar example, the staff assignment needs to be determined as a decision variable, the staff availability is applied as a resource and the promise of bar opening can be considered as a constraint. Thus the problem is interpreted as a constraint satisfaction problem. In the example in the telecommunication network, the antenna can be considered as decision variables which need to be assigned with radio frequencies, the available frequencies are the resources which can be distributed, and finally, the electromagnetic interference laws are the constraints which guarantee the quality of communication service.

By representing them as constraint satisfaction problems, the above real world problems are formally expressed by a set of variables, domains and constraints and many resolution techniques can be adopted to solve this kind of problem which consist in satisfying the constraints while assigning the resources from the domains to the variables.

As mentioned before, the problem model may be unsatisfiable due to lack of resources. In that case the problem is over-constrained and one objective may be to satisfy the maximum number of constraints instead of satisfying all constraints. Another crucial interest of this kind of problems is to find out the reason for failure which causes the unsatisfiability of the problem. Following the terminology proposed by Chinneck [1], such reasons can be interpreted mathematically by an Irreducible Infeasible Subset.

At the beginning of the 1980's in last century [2], the researchers from the linear programming community worked on identifying the satisfiability of the linear programming models. A significant research result from this is the identification of a subsystem inside unsatisfiable linear programming system, which is unsatisfiable. By locating a smaller size of unsatisfiable subsystem, the unsatisfiable reason may be concluded to a comprehensive scale. Following the terminology defined by Chinneck, such subsystem can be named as an Infeasible Subset (Subsystem), the optimal definition of Infeasible Subset is called Irreducible Infeasible Subset (IIS). The meaning is that the reduction of that Infeasible Subset by removing one constraint or variable transforms the subset into a feasible one.

Beside the IIS identification in linear programming, the study also extended to solving SAT (boolean SATisfiability problems). The SAT community heavily studies the MUC/MUS (Minimal Unsatisfiable Core/Subformula, the analogous of IIS in SAT problems) detection in the recent decade and develops techniques which efficiently verify the SAT system unsatisfiability.

In SAT problems, the MUC/MUS consists of a subset of clauses and the implied literals under such clauses. Under the scope of more generalized constraint satisfaction problems, an IIS consists of a subset of constraints and a subset of variables. It is unsatisfiable by itself [1], and impacts on modeling and reasoning about the real world problems cannot be over stated but its practical and theoretical importance are highlighted in both Operational Research and Artificial Intelligence communities.

In the bar opening example, the negotiation between the two people or the different objectives can be discussed when the failure reason is located. Further action may be to improve the current conflict situation. The identification of IIS not only provides a failure reason as the reference for the further actions, but also can be considered as a proof of the problem unsatisfiability. In the more complex example of frequency assignment to a telecommunication network discussed above, the unsatisfiability of a regional telecommunication network may be difficult to prove. After the identification of one IIS inside the problem, the problem unsatisfiability can be proved through the unsatisfiability proof of a relatively smaller size IIS which constitutes a sub-network of the global network.

As mentioned above, numerous resolution techniques were proposed to identify MUC/MUS in SAT problem or IIS in CSP problems during the last decade. These techniques are dedicated to various applications and efficient in particular domains. In this dissertation, a new method dedicated to the Frequency Assignment Problem, named *LCV* (Locator, Constructor and Verificator), is proposed to identify a service blockage zone (interpreted by IIS) in telecommunication network. Also the method is adopted to identify the critical subgraph, which is the IIS in the context of the k-coloring problem.

This dissertation is organized as following. It begins with Chapter 1, the general introduction of Constraint Satisfaction Problems modeling technique (CSP). The key components of CSP will be addressed and several important definitions and properties will be given. Several CSP resolution techniques developed in recent decades from both operational research and artificial intelligence communities will be illustrated.

The second chapter concentrates on the main topic of this dissertation, the Irreducible Infeasible Subset (IIS). The important definitions of Infeasible Subset (IS) and its optimal form IIS will be explained. The important properties of IIS will be demonstrated and IIS identification will be proven.

Based on these theoretical studies, the practical study on adopting the existing method on the IIS identification in the Frequency Assignment Problem (FAP) will be illustrated and analyzed. It brings motivation in studying a new method dedicated for such application.

Chapter 3 begins with the introduction of the Frequency Assignment Problem and how it can be formulated as a Constraint Satisfaction Problem. The different topologies of *CELAR* and *ROADEF2001* benchmarks used to evaluate the performance of the proposed method are illustrated right after the problem introduction. A comparative analysis between the method proposed by Galinier and Hertz [3] and the new method *LCV* will be given to demonstrate the performance leverage carried by *LCV* on the FAP instances.

With the experience in the Frequency Assignment Problem, Chapter 4 attempts to extend the general proposed *LCV* in identifying the critical subgraph in the k-coloring problem. The chapter begins with the definition of the k-coloring

problem and the famous DIMACS benchmarks. The comparison between the state of the art method and the general *LCV* is illustrated by careful analysis and experimental results.

The general conclusion will be given at the end of this dissertation as the essential report of previous study and experience and the perspective attempt to explore new research area in IIS detection.

Chapter 1

Constraint satisfaction problems and resolution techniques

The Constraint Satisfaction Problem is a convenient modeling technique to model many real world applications. For example, assigning the frequencies on an antennae, allocating staff for airline crews, etc. Given a close look at these real world applications, there are several common characteristics with them. These problems consist in assigning the resources, the available frequencies or the human resources, on the decision variables, the antennae or crew scheduling, and respecting the constraints, the electromagnetic interferences or staff availability.

A constraint satisfaction problem consists of three main components: variables, domains and constraints. In this chapter, the important definitions, theorems and resolution techniques concerning constraint satisfaction problem will be presented.

1.1 Definitions for CSP

A constraint satisfaction problem (CSP) consists in finding a legal assignment of values from the domains of variables which satisfies all the constraints of the problem. The variables of CSP is defined as:

Definition 1 (Variable). A variable of CSP is a decision variable x which is waiting to be assigned with a value.

The constraint in CSP can be defined as:

Definition 2 (Constraint). A constraint $c \in C$ is a relation R_c defined on a subset of variables $X(c) \subseteq X$, it regulates the legal combination of values on $X(c)$.

Where $X(c)$ is the subset of variables under the constraint c, respectively the subset of constraints imposed on same variable x is denoted as $C(x)$, the relation R_c defines the permitted Cartesian product of values on the subset of variables $X(c)$. Following the terminology of arity of the mathematical relation, the arity of a constraint can be described as:

Definition 3 (Arity of constraint). The arity of a constraint c indicates the number of variables involved in such a constraint, denoted $|X(c)|$.

The un-ary constraint is such a constraint with arity of 1, the binary constraint is such a constraint with arity of 2 and the n-ary constraint is such a constraint with an arity of n, where $n > 2$. The domain of each variable is defined as:

Definition 4 (Domain). A domain of variable x, denoted $D(x)$, consists of a set of values which is available for assignment on variable x.

With the definition of variables, constraints and domains of variables, a CSP can be essentially described as:

Definition 5 (Constraint Satisfaction Problem). A CSP is a 3-tuple $P = (X, C, D)$, where X is a set of variables, D is a corresponding set of domains, C is a set of constraints applied on X.

Let X' be a subset of X denoted $X' \subseteq X$, $C(X')$ be a subset of constraints imposing exclusively the variables of the subset X'; or let C' be the subset of constraints where $C' \subseteq C$, $X(C') \subseteq X$ be the subset of variables connected by constraints of the subset C'. A subproblem of a CSP can be presented as:

Definition 6 (Subproblem). A subproblem of a CSP is a tuple $P' = (X', C', D)$, where $C' = C(X')$ and $C' \subseteq C$ and $X' = X(C')$ and $X' \subseteq X$.

An assignment $\mathcal{A}$ of a CSP can be written as:

Definition 7 (Assignment and Partial assignment). An assignment $\mathcal{A} = \{a_1, a_2, \ldots, a_m\}$, is a result obtained by assigning one value $a_i \in D_i$ to each variable $x_i \in X_m$, where X_m is a subset of variables in X, denoted $X_m \subseteq X$. In case $X_m \subset X$, the assignment $\mathcal{A}$ is a partial assignment.

Solving a CSP is to find a solution:

Definition 8 (Solution and Partial solution). A solution S of a CSP is an assignment which satisfies all the constraints of this CSP. Respectively, a partial solution is a partial consistent assignment on X' which satisfies all the constraints exclusively on a subset of variables X', where $X' \subset X$.

In both cases, the solution or partial solution, we say that the assignment is valid or consistent. If the assignment does not satisfy all the constraints, it is invalid or inconsistent. Such assignment is not a solution of the CSP. A satisfiable CSP is defined as:

Definition 9 (Satisfiable CSP). A CSP P is called satisfiable if and only if it exists at least one solution to this CSP, otherwise it is unsatisfiable or over-constrained.

For over-constrained or unsatisfiable problems, it is also interesting to find out a compromised solution which respects the maximal number of constraints. Freuder and Wallace declared such problem definition as Partial CSP or Maximal Constraint Satisfaction Problem (Max-CSP) [4]:

Definition 10 (Max-CSP). A Max-CSP is an optimization problem for which the objective is to find one of the assignments that satisfy the maximal number of constraints.

Then we can define the following:

Definition 11 (Optimal solution of Max-CSP). An optimal solution of a Max-CSP is an assignment such that it does not exist another assignment which satisfies more constraints, strictly.

There is a particular category of CSP which cannot be ignored. The boolean SATisfiability problem (SAT) plays an important role in the study of CSP, it is a reference benchmark. It is commonly expressed as a propositional formula which can be described as:

Definition 12 (Propositional formula). A propositional formula $\mathcal{F}$ in Conjunctive Normal Form (CNF) is the conjunction of clauses from a set $C = \{c_1, \ldots, c_m\}$ involving the variable set $X = \{x_1, \ldots, x_n\}$. Each clause $c_i \in C$ is the disjunction of literals from a set L_i. Each literal is either a variable $x \in X$ or its negative $\neg x$. A CNF formula can thus be expressed in the following way:

$$\mathcal{F} = \bigwedge_{i=1}^{m} \left(\bigvee_{l \in L_i} l \right) \tag{1.1}$$

A formula $\mathcal{F}$ is satisfiable if and only if there exists a truth assignment that satisfies all its clauses. The SAT consists in finding a truth assignment for all the variables, such that the formula $\mathcal{F}$ is evaluated to be true, or showing that no such assignment exist. As a particular case of CSP, the SAT inherits the properties of CSP. The clauses of SAT are the constraints of CSP, the variables have the same definition of the variables of CSP and the available values for each literal are two boolean values - *true* or *false*. As CSP with MaxCSP, the MaxSAT is the analogous in SAT problem.

In following sections, the three major techniques used to deal with CSP will be presented. They are constraint propagation, learning system and resolution methods of CSP.

1.2 Local consistency

1.2.1 Definitions of consistency

The consistency or the satisfiability of an assignment refers to the constraints satisfaction or truth of the assignment.

Definition 13 (Local consistency). Local consistency concentrates the conditions for the consistency of subsets of variables or constraints. They require that a partial solution can be extended to an assignment of another variable such that the resulting assignment is consistent.

There are several levels of local consistency, mainly the Node-Consistency, the Arc-Consistency, the Path-Consistency.

Node-Consistency only concerns a single variable, it may be considered as the single variable domain reduction. Thus it can be directly interpreted as the domain definition of the CSP.

Definition 14 (Node-Consistency). A variable $x_i \in X$ is Node-Consistent if any value $a_i \in D(x_i)$ is allowed by the unary constraints on x_i.

Mackworth [5] clearly defined the Arc-Consistency for binary CSP:

Definition 15 (Arc-Consistency). Given two variables $x_i, x_j \in X$ under the same binary constraint c_{ij}. x_i and x_j are Arc-Consistent if for any value $a_i \in D(x_i)$, there exists a value $a_j \in D(x_j)$, such that the values pair (a_i, a_j) is allowed by the binary constraint c_{ij} on x_i, x_j. The value a_j is called a supporter of the value a_i and vice-versa.

The CSP itself is said Arc-Consistent if any couple of adjacent variables is Arc-Consistent. Montanari [6] proposed Path-Consistency as a necessary condition for the pairs of values in binary CSP:

Definition 16 (Path-Consistency). A variable pair (x_i, x_j) is Path-Consistent if there exists an Arc-Consistent values pair (a_i, a_j) satisfying all constraints forming the path between the variables x_i and x_j, where $a_i \in D(x_i)$ and $a_j \in D(x_j)$.

Freuder generalized the notion of local consistencies to k-Consistency [7]:

Definition 17 (k-Consistency). A CSP is k-Consistent if for any $k-1$ variables $(x_1, x_2, \ldots, x_{k-1})$, there exists a consistent partial assignment on these $k-1$ variables and a value $a \in D(x_k)$, such that the partial assignment of k variable is consistent.

The k-Consistency generalizes all the local consistency which contains a subset of k variables, in particular, 1-Consistency interprets the Node-Consistency, 2-Consistency interprets the Arc-Consistency and 3-Consistency interprets the Path-Consistency.

In following sections, several local consistency algorithms will be presented and compared by their time and space complexity.

1.2.2 Arc-Consistency algorithms

1.2.2.1 Global view on Arc-Consistency algorithms

Arc-Consistency (AC) algorithms are local consistency checking algorithms acting as variable domain reduction. The main difference between Node-Consistency and Arc-Consistency is that Node-Consistency exclusively deals with a single variable's domain and Arc-Consistency deals with a pair of connected variables x_i and x_j which are under the binary constraint c_{ij}. Bessière[8] concludes two important reasons to study Arc-Consistency: it is the basic mechanism used in all solvers and the improvement made for Arc-Consistency can be applied to achieve other local consistencies algorithms.

Briefly, all Arc-Consistency algorithms can be described as removing any value from a variable's domain which is Arc-Inconsistent. We use a graph coloring problem to demonstrate the AC verification. In Figure 1.1, node x_i connects with both nodes x_j and x_k, each node has its proper domain, $\{red, blue, green\}$ for nodes x_i and x_k, $\{red\}$ for x_j.

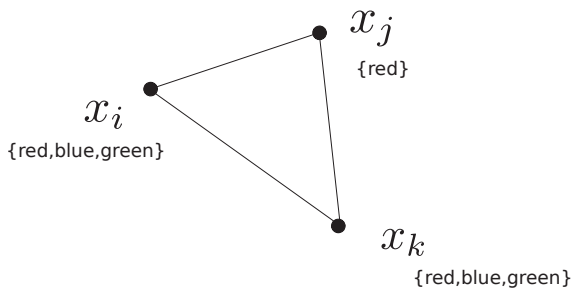

Fig. 1.1: AC on graph coloring problem

The color red in x_i has its supporter in node x_k, which is $blue$ or $green$. But it has no supporter in x_j's domain. With the Arc-Consistency checking, the color red will be pruned from x_i's domain.

Algorithm 1 illustrates the general routine of AC algorithms. The algorithm consists of a Q list which stores the variables waiting for verification. When the domain of a variable x_i is reduced, the constraint connected or neighbor

variables $N(x_i)$ of such variables are inserted into the Q list. Such mechanism avoids the repetitive variables and constraints checking and guarantees that the consistency will be still held while the domain reduction happens. Given a CSP $P = (X, D, C)$, the AC algorithm either returns a CSP P' with a set of reduced domains respecting Arc-Consistent $P' = (X, D', C)$ where $D' \subseteq D$, or determines that the problem is not Arc-Consistent.

Algorithm 1: AC algorithm general routine

Input: A CSP $P = (X, C, D)$
Output: Return an Arc-Consistent $P' = (X, C, D')$ or P is not Arc-Consistent

```
1  Set Q ← V;
2  while Q ≠ ∅ do
3      Pick a variable x ∈ Q;
4      foreach a ∈ D(x) do
5          Verify if a has a support value in each neighbor domain D(N(x));
6          if a has no support in one of these domains then
7              Remove a from x domain D(x);
8          end
9      end
10     if D(x) = ∅ then return P is not arc-consistent;
11     if x has any value a removed from D(x) then
12         Set Q ← Q ∪ N(x);
13     end
14 end
15 return P' = (X, C, D');
```

There are many AC algorithms proposed in the literature, their slight differences are the various implementations in Line 5 of Algorithm 1. In the following sections, several well known AC algorithms will be essentially described in two categories: *coarse-grained* and *fine-grained* which are defined by Zhang and Yap [9].

a) Fine-grained: AC4 and AC6

Classified as fine-grained algorithms, the *AC4* [10] and *AC6* [11] both remember the supporter values for each value in the domain of variables. In case of *AC4*, all the supporter values for each value are recorded with a counter of the number of supporters per value. The generation of supporters is done in a pre-processing step. The advantage of such an approach is that the constraint checking is only executed once during pre-processing, it avoids the checking overhead if the constraints checking is through variable assignment. During the procedure, the value will be removed from the domain only if its supporters counter is reduced to zero. Slightly different from *AC4*, the *AC6* abandons the supporters counter and only records the first consistent supporter value for each value. By recording only one supporter per value, it greatly reduces the space complexity while it may have the risk of increasing the number of constraints

checking since the number of the available supporters values are unknown. *AC6* elegantly avoids such this risk by fixing the values ordering in the domain, thus the values set before the recorded supporter value are inconsistent as a supporter. If the supporter is pruned from the domain, only the values behind it have to be verified and *AC6* will choose the first consistent value as the new supporter.

b) Coarse-grained: AC3 and AC2001/AC3.1

Instead of recording all the supporter values, the coarse-grained algorithms only adapt the constraint checking by brute force. The routine of *AC3* [5] can be exactly described as Algorithm 1. The algorithm verifies all the values of a variable to see if there is at least one supporter value in every adjacent (or neighborhood) variables' domain. If a value of x_i has no supporter in one adjacent variable's domain, it will be pruned from the domain $D(x_i)$. As such value may be a supporter value, the adjacent variables' consistency will be re-verified. The Q list is implemented to avoid the repetitive checking of variables by only adding the variables which are not in the list. It is noticed that checking the values for supporters always begins at the first value of the adjacent variable domain. *AC2001* [9, 12] avoids such repetitive supporter value checking by ordering the values in the domain and recording a pointer to the first supporter's position in the domain. When a supporter is pruned, the new supporter will be found only after such supporter in the domain.

The AC algorithms usually act as the constraint propagator and are embedded in the CSP solvers. In order to choose the right one to integrate inside solvers, it is necessary to compare the algorithms and to wisely implement them in an efficient manner. Let n be the number of variables, e be the number of constraints and d be the maximal size of the domain. The time and space complexities of above four AC algorithms are illustrated in Table 1.1.

Complexity	AC3	AC4	AC6	AC2001
Time	$O(ed^3)$	$O(ed^2)$	$O(ed^2)$	$O(ed^2)$
Space	$O(e)$	$O(ed^2)$	$O(ed)$	$O(ed)$

Table 1.1: Time and space complexity of AC algorithms

In Table 1.1, it is noticed that *AC3* has the highest time complexity while other algorithms are similar in time complexity. The *AC3* and *AC2001* have less space complexity than the *fine-grained* ones, notably *AC4* and *AC6*. Beside above four AC algorithms, there are also several other AC algorithms existing in the literature. The reader can refer to [8], [13] and [14] for more details.

Alongside binary constraints, the AC algorithms can be also extended to deal with high arity constraints. Mackworth [15] proposes a general (multi-arity constraints) AC algorithm *GAC3* based on the binary *AC3* which has a time complexity of $\mathcal{O}\left(er^3d^{r+1}\right)$ and a space complexity of $\mathcal{O}\left(er\right)$, where r is the greatest arity among all constraints.

1.2.2.2 Recursive implementation of *AC3*

Despite of the time complexity of *AC3*, its space complexity is less than both the *fine-grained* algorithms and *AC2001*. Wallace [16] points out that *AC3* computing performance in practice is better than the *fine-grained* algorithms in their experiments. This evidence suggests that *AC3* is an ideal constraint propagator embedded in CSP solvers due to its simplicity and its acceptable worst case computational performance in practice.

Algorithm 2: recursive implementation of *AC3*

Input: A CSP $P = (X, D, C)$
Output: Return P is Arc-Inconsistent or an Arc-Consistent $P' = (X, C, D')$

```
1  foreach x ∈ X do
2      ArcConsistent (x) ;
3  end
   /* ArcConsistent() function                                          */
4  ArcConsistent (x)
5  {
6  Removed ← false;
7  foreach a ∈ D(x) do
8      foreach y ∈ N(x) do
9          if a has no supporter in D(y) then
10             remove a from D(x);
11             Removed ← true;
12         end
13     end
14 end
15 if Removed = true then
16     if D(x) = ∅ then return P is not Arc-Consistent;
17     foreach y ∈ N(x) do
18         ArcConsistent (y)
19     end
20 end
21 }
```

In [14], the authors present a recursive implementation of the AC algorithms which can be described in Algorithm 2. The space complexity is reduced thanks to the recursive adoption of *AC3* algorithm. The major drawback of recursive approach is that it may cause the repetitive variable checking due to the lack of Q list. Our experiments conducted on the CELAR benchmark (see Section 3.2.1 for detail information of benchmark) allow us to respond to this concern,

it shows an average computational time gain of 68.51% on the non Arc-Consistent instances and an average gain of 31.29% on the Arc-Consistent instances.

1.2.3 Other local consistency algorithms

Alongside the Arc-Consistency algorithms, there are also other local consistency algorithms existing in the literature. Based on local consistency enforcement, these algorithms can be divided into two categories:

- The local consistency algorithms stronger than Arc-Consistency.
- The local consistency algorithms weaker than Arc-Consistency.

Under the first category, there are Restricted Path Consistency (*RPC*) [17], Path Inverse Consistency (*PIC*) [18] and Max-Restricted Path Consistency (*maxRPC*) [19] algorithms.

In contrast to Arc-Consistency acting on the domain reduction, these algorithms do not remove the inconsistent value from its domain. Instead, they only record the consistent tuples of values under the form of constraints or partial assignments which may be extended to a solution.

Given a Arc-Consistent instance and two Arc-Consistent variables x_i and x_j, *RPC* guarantees that any unique Arc-Consistent values pair (a_i, a_j) has a value $a_k \in D(x_k)$ which holds the Arc-Consistency on (a_i, a_k) and (a_k, a_j) separately, where x_k is connected with both x_i and x_j.

PIC restricts that, for any variable x_i, there exists at least a pair of variables $x_j, x_k \in X$, such that (a_i, a_j, a_k) are locally consistent for any $a_i \in D(x_i)$ where $a_i \in D(x_i)$, $a_j \in D(x_j)$ and $a_k \in D(x_k)$.

maxRPC respects not only *PIC* condition where (a_i, a_j, a_k) is locally consistent, but also the Arc-Consistency on (a_i, a_j) or (a_i, a_k).

The Table 1.2 shows the time and space complexity of different path consistency algorithms, where e is the number of constraints, n is the number of variables, d is the maximum size of the domain and t is the number of triple of variable (i, j, k) with $c_{ij}, c_{ik}, c_{jk} \in C$. Comparing with the Arc-Consistency algorithms, these algorithms have relatively higher time complexity. Thus in practice, these algorithms may not be ideal candidates to act as an embedded constraint propagator in a CSP solver [8].

Under the category of the local consistency weaker than Arc-Consistency, it includes Directional Arc-Consistency (*DAC*) [21], Full Look-ahead (*FL*), Partial Look-ahead (*PL*) [22] and Forward-Checking (*FC*) [22] algorithms.

Algorithms	time	space
RPC1 [17]	$\mathcal{O}\left(end^3\right)$	$\mathcal{O}\left(ed(n+d)\right)$
RPC2 [19]	$\mathcal{O}\left(end^2\right)$	$\mathcal{O}\left(end\right)$
maxRPC [19]	$\mathcal{O}\left(en + ed^2 + td^3\right)$	$\mathcal{O}\left(end\right)$
PIC1 [18]	$\mathcal{O}\left(en^2d^3\right)$	$\mathcal{O}\left(ed + td\right)$
PIC2 [20]	$\mathcal{O}\left(en + ed^2 + td^3\right)$	$\mathcal{O}\left(ed + td\right)$

Table 1.2: Time and space complexities of several Path Consistency algorithms

DAC maintains the Arc-Consistency between any two connected variables x_i and x_j by ordering $x_i \prec x_j$. It means that all the values authorized in the domain of variable x_i need to find at least a supporter in domain of x_j, but not necessarily for the values in the domain of x_j.

Given a variable ordering $\prec\ (x_1, x_2, \ldots, x_i, \ldots, x_n)$ of n variables and a partial assignment $\mathcal{A}_i$ on the subset variables $\{x_1, x_2, \ldots, x_i\}$, *FC* filters the domains of non-assigned variables $\{x_{i+1}, \ldots, x_n\}$ by removing the values which are not consistent with the partial consistent assignment $\mathcal{A}_i$.

FL and *PL* offer stronger local consistency based on the pre-processing of *FC*. Given the same conditions of *FC* mentioned above, *PL* applies *DAC* on variables subset $\{x_{i+1}, \ldots, x_n\}$ by the defined ordering. Based on *PL*, *FL* even enforces the *DAC* from the variable $x_j \in \{x_{i+1}, \ldots, x_n\}$ to the variable $x_k \in \{x_1, \ldots, x_i\}$.

All these algorithms with local consistency weaker than AC have the same time complexity of $\mathcal{O}\left(ed^2\right)$. Figure 1.2 illustrates the relationship among each local consistencies. The direction of the arrow indicates the relation from the stronger consistency to the weaker consistency.

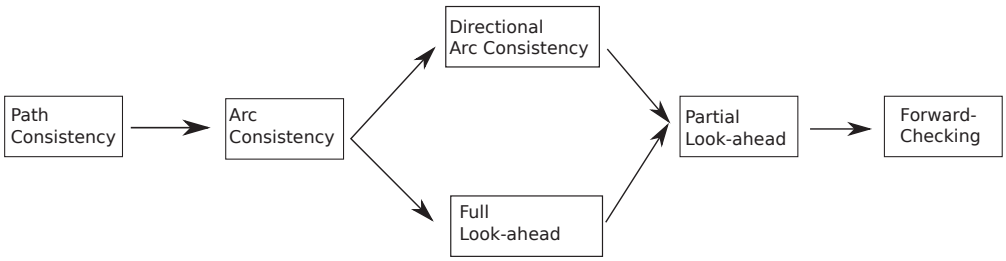

Fig. 1.2: Local consistencies

1.3 *Deadend* learning methods

Deadend learning is a search space reduction technique in solving CSP. Frost and Dechter [23] report several frequently used learning techniques. Before the description of the learning techniques, two basic definitions need to be declared, they are *Deadend* and *Nogood*.

Definition 18 (*Deadend*). A *deadend* is a state of a search node, it occurs when a partial consistent assignment cannot be extended on one variable x. Such variable x is called *deadend* variable.

A *nogood* is defined as:

Definition 19 (*Nogood*). A *nogood* is a pair $(\mathcal{A}_i, X_i)$, where $\mathcal{A}_i$ is a partial assignment on the subset of variables $X_i \subseteq X$, such that no solution of the CSP contains $\mathcal{A}_i$.

A *nogood* is visible when a *deadend* occurs. Theoretically, the whole partial assignment causing the *deadend* can be considered as a *nogood*. The size of such partial assignment may be very large, which adds the space complexity to store it and the time complexity to verify its appearance during search. A minimal *nogood* is that for which by unassigning one of its variable, it is not a *nogood* anymore. Thus the *deadend* learning techniques are required to effectively and efficiently generate the *nogood*. Frost and Dechter[23] compared four types of *deadend* learning techniques:

- Value-based shallow learning.
- Graph-based shallow learning.
- Jump-back learning.
- Deep learning.

Example 1 is used to demonstrate above learning techniques:

Example 1. Given a CSP $P = (X, D, C)$, a variable ordering $\prec (x_1, x_2, \ldots, x_5)$ and a partial consistent assignment $\mathcal{A}_{1\ldots4} = \{a_1, a_2, a_3, a_4\}$ on variables subset $\{x_1, x_2, x_3, x_4\}$, a *deadend* occurs when the search attempts to find a consistent value on x_5 (see Figure 1.3).

a) *Value-based learning*

Suppose in Example 1 that the value a_3 is consistent with all the values in the domain of the variable x_5, while each of the other values of $D(x_3)$ is inconsistent with several values in $D(x_5)$, and together, the values (a_1, a_2, a_4) are

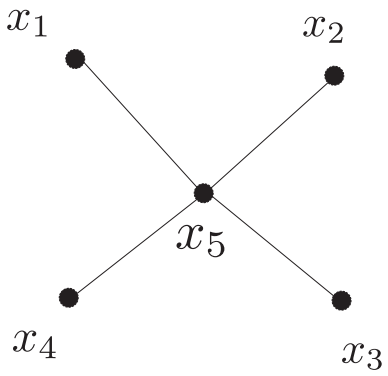

Fig. 1.3: Graph of Example 1

inconsistent with all values in $D(x_5)$. Since the value a_3 is irrelevant in inconsistency, it can be removed from the *nogood* by *Value-based learning*. The *nogood* becomes (a_1, a_2, a_4) on the subset (x_1, x_2, x_4). The time complexity of *Value-based learning* is $\mathcal{O}(n)$ at each *deadend*.

b) *Graph-based learning*

Differing from *Value-based learning*, *Graph-based learning* will judge the relevance of the *nogood* by the connectivity among the variables. Since all the variables (x_1, x_2, x_3, x_4) are connected with x_5, the *nogood* is (a_1, a_2, a_3, a_4) on variables (x_1, x_2, x_3, x_4). The complexity of learning at each *deadend* is $\mathcal{O}(n)$.

c) *Jump-back learning*

During the search of Backjumping algorithm [24], it maintains a conflict set for the variable involved in instantiation. If such a variable is a *deadend* variable as x_5, the conflict set is a *nogood* for the problem. The complexity of this learning method is constant during the Backjumping search.

d) *Deep learning*

Deep learning literally analyzes the inconsistency between the permutation of values from a partial consistent assignment and the values in the domain of the *deadend* variable. Suppose in Example 1 that (x_3, a_3) does not conflict with any values in the current variable x_5, (x_1, a_1) conflicts with the values subset I_1 of variable x_5, (x_2, a_2) conflicts with the values subset I_2 of variable x_5, (x_4, a_4) conflicts with the values subset I_4 of variable x_5, where $I_1 \neq I_4$,$I_1 \neq I_2$

and $D(x_5) = I_1 \cup I_2 = I_1 \cup I_4$. Two values permutations $(x_1, a_1; x_2, a_2)$ and $(x_1, a_1; x_4, a_4)$ are recorded as *nogood*. Dechter[25] pointed out that its cost is exponential by the size of the initial conflict set.

Frost and Dechter mention that these learning techniques are very costly in practice. The techniques are only efficient when the *deadend* occurs frequently during the search [23]. An effective objective of learning is to minimize the size of the *nogoods*, in other words, to minimize the number of variable/value pairs forming a *nogood*. This phenomenon can be explained with the lattice in group theory. Since the *nogood* represents the property of all combinations in the search space, the smaller the property size is, the more combinations are represented. Thus the smaller size nogood can cut larger space in the search space.

Katsirelos and Bacchus [26] suggest that an exponential number of *nogoods* and high arity of *nogood* have a negative impact on the performance of *deadend* learning techniques. Thanks to clause learning in the SAT problem, they proposed to generalize the *nogood* under the form of learned clauses of SAT. They concluded that even if such approach has a positive impact on solving CSP instance, it is still not effective in the general case.

One significant problem of *nogood* learning is the number of *nogood* recorded. The constantly increasing number of *nogoods* raises the time complexity of *nogood* matching and space complexity of *nogood* storing. Relevance-bounded and size-bounded methods are two popular techniques to limit the number of *nogoods* recorded. The relevance-bounded technique eliminates the *irrelevant nogood* which are defined by the number of common variables inside *nogood* and appearing in the current assignment. The size-bounded technique records exclusively the *nogood* with determined variables size.

Bayardo and Miranker [27] investigate the above *deadend* learning techniques with size-bounded and relevant-bounded learning techniques. They conclude that the proposed relevant-bounded learning technique is more efficient than the size-bounded technique.

From our knowledge, the adoption of learning techniques is not effective in solving general CSP instances. The quantity of generated *nogood* and the arity of *nogood* need to be carefully defined, the constraint propagation techniques need to be integrated with learning procedures to generate the *nogood* efficiently [28]. In the next section, the resolution techniques of CSP will be explained.

1.4 Search techniques for CSP

The search techniques to solve CSP can be loosely divided into two categories: complete search and incomplete search. The methods under the category of complete search explore all consistent solution and can check all solutions. The main technique adopted is the backtracking algorithm. The methods under the category of incomplete search does not explore the whole search space and may only carry out one solution, stochastic local search algorithms are the major solvers to solve the CSP.

1.4.1 Complete search - Backtracking and MAC

Backtracking is the primary complete search algorithm for CSP, it explores the search space based on a partial instantiation in a depth-first manner. The constraints are used to verify whether an extension of variables assignment may lead to a feasible solution. During the search process, all the variables can be classified into three categories:

- Past variables: instantiated variables.
- Current variable: waiting to assign variable.
- Future variables: uninstantiated variables.

A backtracking algorithm consists of two phases: a forward phase and a backward phase. In the forward phase, one of the future variables is selected which is so called the current variable. Thus the current partial solution is extended by assigning a value on the current variable which is consistent with the partial assignment on past variables. The backward phase occurs when there is no existing consistent assignment for the current variable; backtracking returns to the previous last assigned variable in the past variables and try to re-assign another consistent value for such variable (see Figure 1.4).

Maintaining Arc-Consistency during search (*MAC*) [29] is considered the most efficient backtracking method to prove the satisfiability on the general CSP [30]. It embeds an Arc-Consistency propagator inside a chronological backtracking routine. After a value is assigned on the current variable, it will verify if the future variables are Arc-Consistent with the partial consistent assignment. If all the values of one of the future variable's domain are wiped out during Arc-Consistency checking, the current variable will unassign its value and attempt to find another available value in its domain which is consistent with previous partial assignment. If no consistent value can be find, then the backtrack occurs.

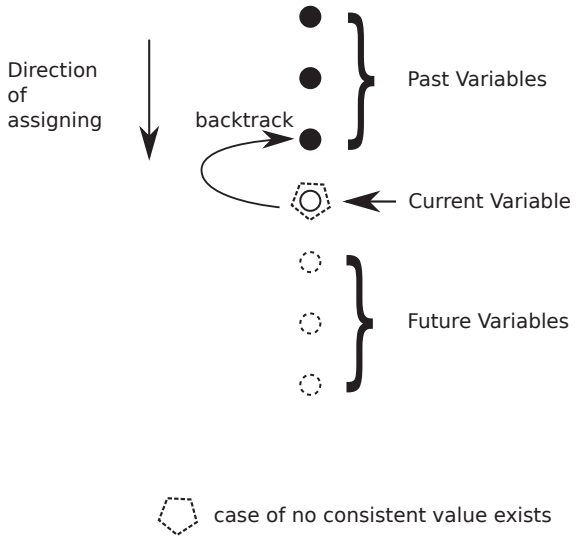

Fig. 1.4: Backtracking process

The major advantage of the *MAC* algorithm is that all the future variables' domains are verified and reduced based on the inference of the partial assignment on past variables. It dramatically reduces the search space in which there will be no existence of feasible solutions containing the partial assignment. The longer the partial assignment is extended, the more the search space is reduced.

There exists many implementations of *MAC*, here we propose a simple way to leverage the Arc-Consistency checking. Instead of applying directly on all the variables, it is wiser to apply the *Forward-Checking* on the future variables on the basis of the current partial consistent assignment before applying Arc-Consistency algorithm on future variables exclusively.

Three approaches can be used to guide the process in an efficient way:

- Choose the lightweight coarse-grained AC algorithms.
- Use cache during search.
- Reduce space and time complexity while implementing *AC* algorithms in *MAC*.

The advantage of applying coarse-grained AC algorithms (like *AC3*, *AC2001/3.1*) is to avoid the maintenance of a heavy and complex information database during search, then it can speed up the propagation process in general. Van Dongen compared *AC2001/3.1* version of *MAC* and his *AC*3_d[31] version of *MAC* [32], he concluded that the *AC*3_d version of *MAC* is a lightweight *MAC* which is ideal for a cheap constraint check problem. He showed that the *AC*3_d version of *MAC* keeps the space complexity of $O(e + nd)$, while *AC2001* version of *MAC* has a space complexity of $O(ed\min(n, d))$, where $\min(n, d)$ indicates the smaller number between the number of variables n and the maximum

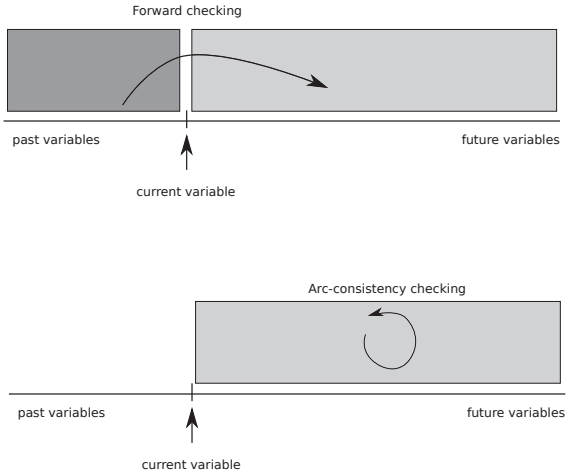

Fig. 1.5: MAC implementation

size of the domains d. In [13], Mehta and van Dongen propose two variants of *AC*3_d - $AC3_{dl}$ and $AC3_{ds}$. They conclude that the $AC3_{dl}$ has a better equilibrium between the memory space for recording supporter values and the check saving.

Recently, Likitvivatanavong et al.[33] give us the detailed implementations of *AC3* and *AC2001/3.1* in *MAC* version. They propose the cache technique to increase the efficiency of the MAC algorithms. They report the speed up of *MAC3* by 30% on hard problems.

Reducing space complexity is recently introduced by Régin[34]. He proposes some improvements of *MAC6* (based on *AC6*) and *MAC7* (based on *AC7*) which keep the space complexity of the original Arc-Consistency algorithms.

Alongside with *MAC*, the Backjumping method [24] including Graph-based Backjumping (*GBJ*), Conflict-directed Backjumping (*CBJ*), and Dynamic Backtracking (*DBT*) [35, 36] are also frequently adopted in solving CSP.

1.4.2 Incomplete search - Heuristics

The 3-SAT problem is one of Richard M. Karp's 21 classic NP-complete problems [37]. Its more general form as CSP is not tractable due to its complexity. Regarding the sizes of real world problems, heuristics based resolution techniques are more favorable as a compromise between solution quality and computational time.

Tracing the history between the artificial intelligence community and the operational research community, many heuristics or meta-heuristics were proposed in the context of different applications. Local search algorithms play an important role in solving constraint satisfaction problems [38] thanks to their efficiency and effectiveness. As

the numerous proposition of heuristics during the recent decades, we essentially intend to address several important heuristic designs in the following text.

Variable Ordering [39, 40] and Value Ordering [21] heuristics are widely chosen to accelerate the search by specifying the critical search space. Several Values Ordering heuristics in the literature are:

- *Bayesian Networks* based solution estimation [41], with the spanning tree decomposition of a CSP, the probabilities on the values of variables are generated inherently.
- *Look-Ahead* based values selection [42], it makes the assumption on the values of the non-assigned variables with maximal number of supporter values, which have more promise to be extended to a solution.
- *Nogood learning* based values selection [43], it judges the extensibility of values by the *nogood* learned during the search.

Variable Ordering can be roughly divided into two categories: *Promise* and *Fail-firstness* [39]. *Promise* guides the search toward the promise search space which may contain feasible solutions, while the latter cuts the search space by telling the *deadend* variables to the search. In [44], the author gave a more comprehensive study on the performance of various Variable Ordering heuristics.

MinConflict [45] is the primary technique in designing heuristics. This technique consists in assigning a value on a new variable with *Minimum Remaining domain Values* (MRV). This mechanism attempts to reach a *deadend* at the high level of the search tree, further to cut the search space with no feasible solutions inside. That is the reason such an approach is also called *fail-first*, which always assigns next *the most constrained* variable.

Another strategy is to measure the next variable to assign by its degree which is the number of constraints implied on it. Such an indicator gives a structure representation of *the most constrained* variables. Brelaz combined a degree and *MRV* heuristic in solving graph coloring problems and reported great success despite its simplicity [46].

Statistic learning is also widely adopted in heuristic design. It associates the variables with the conflict value pairs between each other. Such conflicts can be generated during the search routine, this indicator repeatedly reminds the search to enforce the satisfaction on high score variables. The *Breakout* algorithm proposed by Morris [47] adopts such an approach to indicate the subset of variables which are difficult to be satisfied together.

The successful of integrating local consistency algorithms into backtracking search inspires heuristic design as well. The *CN-Tabu* [48] and *NG-Tabu* [49] both embed an Arc-Consistency algorithm inside Tabu Search and achieved great performance leverage on the Frequency Assignment Problem. With the Arc-Consistency based inference, these algorithms dramatically accelerate search speed by reducing the search space.

As numerous local search techniques are proposed by both the artificial intelligence and operational research communities, the interested readers can also refer a comprehensive study of local searches on CSP resolution in [50].

1.4.3 Techniques in modern SAT solvers

The boolean SATisfiability problem has its particular characteristics which are significantly different from the general form of CSP. There is a particular method of propagation technique named *Unit Propagation* (UP) which is cheaper and more efficient than in case of local consistency algorithms in CSP [38]. Many modern SAT solvers embed the UP in *DPLL* (Davis-Putnam-Logemann-Loveland algorithm [51]) procedure to accelerate the clause assignment during search. Such a technique is applied on the clause whose literals are all determined except one literal. It consists in assigning a *true* value on one literal of the clause whose other literals are assigned with *false*, thus the clause becomes *true*. This technique is widely adopted thanks to its simplicity and effectiveness.

With the enforcement of UP, the learning from clauses can be generated effectively during the *DPLL* procedure. The clause learning can be essentially demonstrated by the following example.

Example 2. Given a SAT instance in form of Conjunctive Normal Form (CNF) which contains three clauses, $\omega_1 : (\neg x_1 \vee x_2 \vee \neg x_3), \omega_2 : (\neg x_1 \vee x_3 \vee x_5), \omega_3 : (\neg x_2 \vee \neg x_3)$.

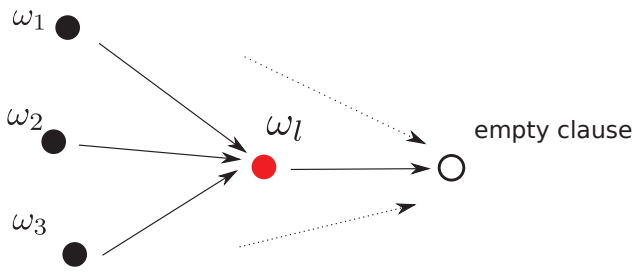

Fig. 1.6: Resolution graph of Example 2

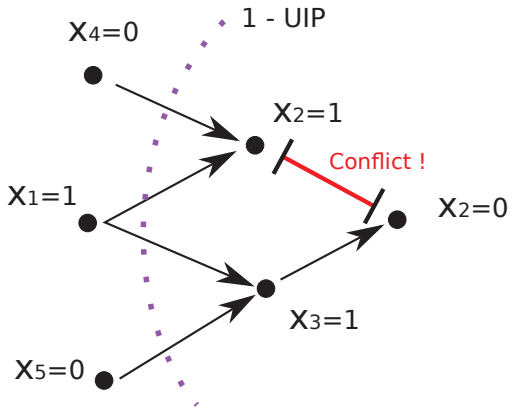

Fig. 1.7: Implication graph of Example2

Regarding the *1-UIP* (Unique Implication Point) learning schema [38], the new clause $\omega_l : (\neg x_1 \vee x_4 \vee x_5)$ will be learned from the previous *DPLL* procedure through the resolution graph of this SAT instance (Figure 1.6). Figure 1.7 shows how the clause ω_l is learned from ω_1, ω_2 and ω_3 by applying UP with a partial assignment $(x_1 : 1, x_4 : 0, x_5 : 0)$. With the given partial assignment, the UP procedure decides the variable x_3 being 1 to keep the clause ω_2 $true$. With x_3 assigning 1 and the clause ω_1, x_2 should be assigned with 1 to keep the clause $true$. While at the same time, the variable x_2 should be 0 to keep the clause ω_3 as $true$. A conflict assignment on x_2 occurs. Then by applying *1-UIP*, the conflict clause $\omega_l : (\neg x_1 \vee x_4 \vee x_5)$ is learned.

The key observation here is that the cost of clause learning during *DPLL* is relatively cheap in SAT problem, and the result of such learning can be smoothly integrated into *DPLL* procedure.

As a chain reaction, the more clauses are learned leads to even more learned clauses. Accompanying with the UP, the satisfiability testing will be greatly accelerated. This is why satisfiability testing in SAT is relatively more efficient and more effective than *deadend* learning in general CSP.

The description of above techniques adopted in SAT shows high efficiency and effectiveness in solving the SAT problem. While it also proves that these techniques are particular in SAT, and they are very difficult to be adopted in general CSP.

This section was dedicated to demonstrate the different characteristics and resolution techniques between SAT and CSP in order to show that the direct adoption of SAT resolution techniques on CSP is difficult to be achieved without significant modifications. For more information on SAT resolution methods, the reader can refer to the recent surveys written by Gu et al. [52], Hirsch [53], Hoos and Stützle [54] and Lynce et al. [55].

1.5 Conclusion

The first section of this chapter introduces the basic definitions of CSP, also including the Max-CSP in case of over-constrained CSP. A particular case of CSP named the SAT problem is also introduced as material needed to understand its resolution techniques.

Also important resolution techniques like Arc-Consistency algorithms and backtracking algorithms are presented and illustrated with their complexity. Several Arc-Consistency algorithms are compared based on their time and space complexity. From such a comparison, we conclude that *AC3* algorithm achieves a better compromise between computational performance and simplicity of implementation. A recursive implementation of *AC3* algorithm is also proposed and shows its performance improvement on over-constrained instances. Among the backtracking algorithms, *MAC* is efficient to provide the unsatisfiability or satisfiability proof by using simple brute force of constraint propagation.

Beside the exact approach to solve the constraint satisfaction problem, there is also the heuristic approach whose methods vary according to different applications.

Along with the algorithm to solve the general form of CSP, the specific resolution techniques devoted to the SAT problem are also presented to demonstrate their efficiency and effectiveness on proving the satisfiability/unsatisfiability of the SAT instances. In next chapter, we will focus on the main topic of this dissertation, the Irreducible Infeasible Subset identification.

Chapter 2

Irreducible infeasible subset (IIS)

In the previous chapter, CSP as a convenient problem modeling technique is introduced. Its definition and several classical resolution techniques are briefly described. This chapter will focus on the primary subject of this dissertation - the Irreducible Infeasible Subset. As already mentioned in the general introduction, it provides an answer to a crucial question: *how we deal with the situation where the given problem is over-constrained?*

Freuder and Wallace answer this question with their proposition of *Partial CSP*, or *Maximal CSP* (Max-CSP). The objective of this model is to propose a solution which respects the maximal number of constraints. The *maximal* dedicates itself on a specific measure objective. Such an objective can be measured by the number of satisfied constraints, the sum of weights of satisfied constraints, etc.

Another possible response is to locate the unsatisfiability reason in an over-constrained CSP. The study on finding such reason can be traced back to van Loon's research paper [56] in 1980's. He begins the studies on finding irreducible inconsistent systems in linear inequality systems. He is followed by Chinneck [57] who develops the methods to deal with infeasible subset in linear and mixed integer programming. Their works is further extended by Gleeson and Ryan [58], Greenberg and Murphy [59] and others.

All this research inspire the studies on finding the infeasible set on SAT instances. Numerous methods have been proposed during the decades to identify the Minimal Unsatisfiable Subformula/Core (MUS/MUC) in SAT which is the Irreducible Infeasible Subset in SAT instances. Based on these studies, the IIS identification is also extended in different applications which can be modeled as constraint satisfaction problem. In this chapter, we will focus on the IIS identification of CSP. It begins with the definitions of IS (Infeasible Subset) and IIS (Irreducible Infeasible Subset). The methods dedicated to the IIS identification in the literature will be classified and illustrated. Finally, the motivation behind this dissertation will also be explained.

2.1 Definitions of infeasible subset, irreducible infeasible subset and critical set

Within an over-constrained or over-determined constraint satisfaction problem, it may exist one or several infeasible subsets which represent the failure reason of finding a consistent complete solution. The definition of such infeasible subset can be described as:

Definition 20 (Infeasible Subset). An Infeasible Subset (IS) is a subproblem of a CSP, for which there is no solution.

Since an over-constrained problem itself can be considered as an IS, which does not bring any contribution in problem solving, a further definition dedicates to problem resolution can be described:

Definition 21 (Irreducible Infeasible Subset). An Irreducible Infeasible Subset (IIS) is an IS which becomes feasible by removing any of its constraints or variables.

The above definition of IIS is based on the destructive view which indicates the state changing based on removing a constraint or variable. The removed element brings the state change from infeasible to feasible. Thus the IIS can be regarded as:

Definition 22 (Irreducible Infeasible Subset of variables). An infeasible set X of variables is said irreducible (IIS) if any proper subset of X is feasible.

Definition 23 (Irreducible Infeasible Subset of constraints). An infeasible set C of constraints is said irreducible (IIS) if any proper subset of C is feasible.

From the minimizing or optimal view, the definition of IIS can also be interpreted as:

Definition 24 (IIS in optimal view). If there is no IS strictly included inside one IS, then such IS is an IIS.

In contrast to the above terminology, in the SAT community the more frequently adopted terms for IS and IIS are Unsatisfiable Core/Subformula (UC/US) and Minimum Unsatisfiable Core/Subformula (MUC/MUS).

A CSP can be conveniently represented as a graph. In the context of graph theory, the IIS holds the following property:

Property 1. An IIS is a connected subgraph of a graph $G = (V, E)$ which defines a CSP. There exists a path composed by the connected constraints between any two variables in IIS.

If an IS is not a connected subgraph, there are potentially smaller IS inside it. By identifying one connected IS inside it, we may take one step forward to an IIS. Along with IS and IIS definitions and their property, the following notation will be introduced to ease the description of resolution techniques:

Definition 25 (Critical constraint). A critical constraint of a CSP $P = (X, C, D)$ is a constraint violated under an assignment which eventually becomes satisfied under another assignment thanks to a local search flip while other constraints become violated [60].

Similarly, a critical variable can be defined as:

Definition 26 (Critical variable). The variable under the critical constraint is called critical variable.

From above definition, all critical constraints cannot be satisfied simultaneously. The critical constraints and the other constraints eventually form an unsatisfiable subproblem of a CSP. It looks similar to the definition of IS (see Definition 20). We consider:

Hypothesis 2.1 *The critical constraints are the candidate constraints to form the IIS.*

Based on the above hypothesis, the IIS identification can be accomplished through identifying the critical constraints of a CSP.

In order to simplify the method description, we define that the constraints subset of one IIS is a critical constraints subset whose unsatisfiability has not yet been proven. Regarding the vocabulary consistency, the terms *critical constraints subset* and *constraints subset of IIS* will be used interchangeable and will be denoted H. In next section, the resolution techniques to identify IS/IIS inside over-constrained or over-determined problems will be presented.

2.2 Resolution techniques in the literature

The contributions carried out by the SAT community cannot be ignored. During recent decades, many methods have been proposed which can be roughly classified into two categories:

- Satisfiability testing approach based on the state change definition (see Definition 21).
- Hitting set approach inspired from the point of view of the hitting set (see Theorem 1).

In following section, we will essentially describe several existing methods by above classification.

2.2.1 Satisfiability testing approach

The significant feature of these methods is an adoption of a satisfiability testing solver which is executed iteratively. On each iteration, it provides an unsatisfiability or satisfiability proof on a subproblem which is constructed (or de-

structed) by inserting (or removing) a constraint in (or from) the current subset of constraints. The satisfiability and unsatisfiability phase transition [61] [62] will be identified during these iterations. The constraint removed or inserted, which leads the transition between unsatisfiability and satisfiability, is defined as transition constraint:

Definition 27 (Transition constraint). In a CSP $P = (X, C, D)$, c being a constraint in C, and C_i being any subset of C, which holds $C_i \subseteq C$ and $c \in C_i$. If C_i is unsatisfiable and $C_i \setminus \{c\}$ is satisfiable, then c is called a transition constraint.

Based on above definition, we also have:

Property 2. Each constraint belonging to an IIS is a transition constraint.

From above definition, it is noticed that the *transition constraint* is the passage from unsatisfiability to satisfiability or satisfiability to unsatisfiability. By identifying such constraints, not necessary an IIS, but an IS can be generated. In order to prove the unsatisfiability, the satisfiability testing is explicitly employed to provide the unsatisfiability or satisfiability proof on a subset of constraints.

In [63] and [64], the authors give the time complexities for both inserting and removing constraints.

- $\mathcal{O}(m)$ for removing constraints.
- $\mathcal{O}(km)$ for inserting constraints.
- $\mathcal{O}(k \log m)$ for inserting constraints with binary search.

Where m is the total number of constraints of the problem and k is the number of constraints belonging to an IS. It is clear that the removing approach is more efficient than the others. For both removing or inserting methods, the satisfiability testing solver runs iteratively during the IIS detection procedure. In order to accelerate the computation, some heuristics may be adopted to propose the potential variables or constraints which are possibly belonging in an IIS. The hybrids include either a collaboration or an integration mode.

In integration mode, the heuristic is charged to propose the next candidate variable or constraint to be processed under the umbrella of the exact satisfiability testing solver. In case of exact tree search algorithm, it verifies the unsatisfiability of the subproblem at the top of the tree search.

In collaboration mode, an heuristic iteratively collects the critical constraints one by one until the heuristic cannot find a partial consistent assignment on such subset of constraints, then the subproblem formed by such constraints subset is injected into an exact solver. If the solver finds a consistent partial solution, the constraints subset will be returned into the heuristic procedure and extended according to the connectivity property.

In following sections, several IIS identification algorithms introduced in recent decades will be essentially described based on integration or collaboration modes.

2.2.1.1 Integration mode

These methods consist in embedding an heuristic in an exact satisfiability solver. The heuristic selects the constraints and either inserts them into a satisfiable subproblem, or removes them from an unsatisfiable subproblem iteratively. The satisfiability solver verifies the unsatisfiability on such extracted subproblem and attempts to detect the transition between satisfiability and unsatisfiability.

Hemery et al. [61] propose an exact satisfiability testing approach to find the IIS in the frequency assignment problem. Under the umbrella of the *MAC* algorithm, the *wcore* heuristically selects the variables to insert in a backtracking search tree according to the violation weights of variables. The violation weights of variables are generated heuristically and indicate the hardness of variables to be satisfied together with existing partial assignment found by *MAC* [65]. Based on this approach, Grégoire et al. [66] improved *wcore* by eliminating the occurrence of backtracks during MAC, which yields better solutions in terms of IIS size and computational time.

In contrast to the above methods, *zMinimal* proposed by Zhang and Malik [67] is based on learning from a resolution graph [68] generated during a *DPLL* [51] procedure to solve SAT instances. The resolution graph represents a directed acyclic graph from the original clauses, through the clauses learned from processed clauses, to the empty clause which indicates a MUS. During the procedure, the non-inference clauses are pruned and only the impliedly learned and original clauses are studied to identify a MUC. Figure 2.1 represents a resolution graph generated during the *DPLL* procedure, the black nodes ω_1, ω_2 and ω_3 represent the original clauses, the red node ω_l represents a learned clause and the empty node is an empty clause. It indicates the relation between the learned clauses and the original clauses. When the empty clause happens, the original clauses induced are identified and are considered as the clauses inside a MUC. The algorithm cannot guarantee a MUC. While if the *zMinimal* is executed iteratively, it may identify a MUC eventually.

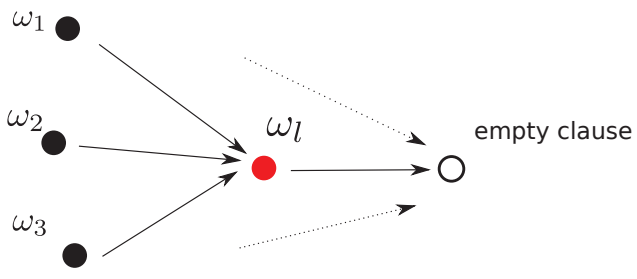

Fig. 2.1: Resolution graph

Zhang and Malik, Oh et al. [69] propose the *AMUSE* algorithm for SAT problem. The algorithm implicitly defines single critical literals of each clause. By doing so, it uses the critical variables in the search tree and enforces the generation of learned clauses.

2.2.1.2 Collaboration mode

The methods in this mode consist in organizing a collaboration between an heuristic and an exact satisfiability solver on the global unsatisfiable problem. The heuristic constitutes one or several subproblems on which we cannot find a partial consistent solution, or on which we assume that there is no consistent partial solution through statistic learning. Then the subproblem is input as an entity into satisfiability testing solver to verify its unsatisfiability.

Eisenberg and Faltings's *BOBT-SUSP* [70] combines Morris's *breakout* algorithm and the backtracking algorithm to identify the IS. The algorithm employs the *breakout* [47] method to identify the IS. During each iteration of the *breakout* algorithm, the weights on violated constraints are incremented. The objective function takes into account the weights to guide the search towards the unsolvable subproblems. Depending on these weights on the violated constraints, the problem is filtered and divided into several subproblems. Then backtracking algorithm is used to verify the unsatisfiability of these subproblems.

Grégoire et al. [60] propose a local search named *AOMUS* to distinguish MUS from the unsatisfiable SAT instances. The algorithm records the subsets of clauses on which a local search [71] fails to find a partial consistent solution. During the search, the hardness of clauses is recorded by the scores. The algorithm iteratively removes the lowest scores clauses from the formula, and records the last approximated unsatisfiable core in a stack until it finds a satisfiable subset of clauses. Thus the algorithm works on adding the last removed clauses into the stack and attempts to verify if it can identify the transition clause. When the subset of clauses in the stack forms an unsatisfiable core, the removal procedure analyzes the scores of the clauses again, and prunes one lowest score clause each time to reduce the size of unsatisfiable core. The procedure is stopped when it reaches again the transition clause. The *Walksat* [72] is employed as the core local search to give an approximated unsatisfiability proof.

Instead of identifying only one critical constraint per iteration, van Maaren and Wieringa's approach consists in finding a bunch of critical constraints per iteration [73]. All processed constraints are sorted by their arities in increasing order. They noted that if the MUC's size is relatively small the exact approach will outperform the heuristic approach.

Banda et al. [74] reduce the computational time by reducing the size of the original problem. They noticed that the transition constraint is strictly inside the IIS and a single IIS holds the connectivity property, while the identification of the transition constraint is quite inconvenient without an efficient satisfiability testing solver.

Mazure et al. [71] adopt an heuristic to record the critical constraints which are violated during the heuristic search, then those literals belonging to critical clauses are input into a branch-and-bound algorithm to verify the unsatisfiability.

2.2.1.3 Summary

The development of the satisfiability testing approach consists in adopting heuristic techniques to either generate the weights/scores of each violation of each constraints for statistical learning during the search, or loosely reduce the size of the subproblem and keep the exact satisfiability solver concentrating on the subproblem which potentially forms an IIS.

The weight/score system adopted in this approach provides important statistic learning material, which either can be considered as the measurement of IIS-variable or IIS-constraints independently or guiding the search towards the hard subproblem by integrating the weights into the objective function. The weight representing statistic learning still plays an important role in indicating the subset critical constraints.

It is always fine that the problem can be reduced without losing the completeness of its IIS, thus the exact proof of unsatisfiability is easier to achieve than working on the entire problem. Derived from above studies and experiences, the general routine of this approach can be described as: a heuristic filters the problem and reduces its size, then an exact method iteratively runs on the proof of unsatisfiability of such subproblem by removing or inserting the constraints.

2.2.2 Hitting set approach

Recently, research has shown the relationship between IIS detection and hitting set problem [75, 76, 3]. The hitting set approach is based on the relation between hitting set problem and IIS detection. The relation is clearly revealed by Bruni from Università di Roma. In his paper [77], Bruni used plain English to describe the relationship and provided the proof.

The significant difference between the hitting set approach and the previous approach is that the Max-CSP solver is heavily employed instead of a satisfiability testing solver. A hitting set problem (also known as graph transversal problem or set covering problem by different communities), is one of the key problems in the combinatorics of finite sets and the theory of diagnosis. The problem is known as NP-complete, and it can be described as follows:

Definition 28 (Hitting set). Given a set M of elements, a collection $L = \{l_1, l_2, \ldots, l_m\}$, such that $l_j \subseteq M$ and $\bigcup l_j = M$. A hitting set is the subset $I \subseteq M$ of elements that hits every set of L, for which $I \cap l_i \neq \emptyset$ for every $l_j \in L$.

Regarding the definition of the hitting set problem, the violated constraints subsets C_{vi} of a CSP can be modeled as the nonempty subsets l_i thus an IIS is the hitting set of the collection of violated constraints subsets. Bruni [77] pointed out that:

Theorem 1. *Given any violated constraints subset C_v of a complete assignment in an over-constrained CSP, and any constraints subset of IIS H, the intersection of these two subsets is not empty, denoted $C_v \cap H \neq \emptyset$.*

The proof of above theorem is given as:

Proof. In order to prove that $C_v \cap H \neq \emptyset$, we can prove that $C_v \cap H = \emptyset$ is wrong.

Given any complete assignment $\mathcal{A}$ on an over-constrained CSP $P = (X, C, D)$, we obtain a violated constraints subset C_v based on such assignment. Under the assumption that the intersection between H and C_v is empty, denoted $C_v \cap H = \emptyset$, we have $H \subseteq C \setminus C_v$. Since all violated constraints are excluded from H, there is no violated constraint in H under the assignment $\mathcal{A}$. This conclusion breaks the IIS definition 20 – there is no consistent partial assignment on the subset constraints H.

Based on the IIS definition, by removing exactly one constraint from each IIS of a CSP, the problem becomes satisfiable. Thus the exact total number of violated constraints in an optimal Max-CSP solution can be determined through the IIS detection. Example 3 gives an illustration of IS detection by the hitting-set approach.

Example 3. To color a graph as shown in Figure 2.2, the numbers represent the indices of the nodes and the letters represent the indices of the arcs. The chromatic number of this graph is $\chi(G) = 3$. The problem is infeasible with only giving 2 colors. A collection L of violated constraints subsets can be described as: $\{a, f\}$, $\{a, g\}$, $\{b, f\}$, $\{b, g\}$, $\{c, f\}$, $\{c, g\}$, $\{d, f\}$, $\{d, g\}$ and $\{e\}$. Two hitting sets of such collection can be described as $\{a, b, c, d, e\}$, $\{f, g, e\}$ which are also two IS (luckily the IIS) of the problem.

From above illustration, two IS are carried out by identifying two hitting sets on a collection of several violated constraints subsets. The collection of violated constraints subsets is generated by iteratively executing a Max-CSP solver. In contrast to the satisfiability testing approach, the Max-CSP solver is heavily employed.

Over-constrained problems, whose unsatisfiabilities are difficult to verify but the Max-CSP solutions can be easily found by Max-CSP solvers, are suitable for such approach. For example, the unsatisfiability of the k-coloring problem

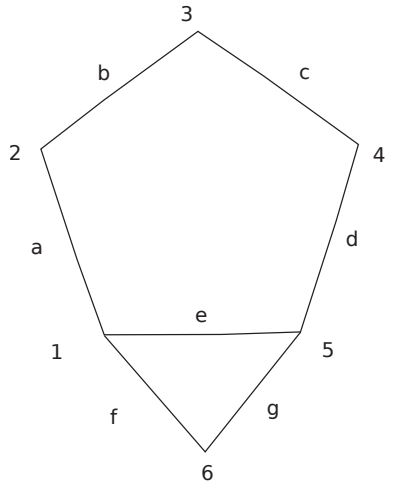

Fig. 2.2: The k-coloring problem of Example 3

is difficult to prove when k is close to the chromatic number. In such a case, the hitting set approach shows its dramatical performance increase [78].

Simply formulating the IIS detection into a hitting set problem will not ease our problem. First, the hitting-set problem itself is NP-complete [37]. If the complexity of the problem cannot be reduced, there is no gain in re-modeling the problem. Secondly, it is costly to enumerate the collection of all violated constraint subsets in different assignments to identify only one IIS.

One idea is to generate a hitting set which has exactly one element in common with each subset in the collection. In case that there is no intersection between any of two subsets, the elements of the hitting set will be exactly one element from each subset. Due to the connectivity property of IIS, all these elements will form a connected component. There are many IIS identification methods in the literature developed by following the above hitting set approach. In the remaining part of this section, we will describe several of these methods.

Bruni and Sassano [75, 77] proposed an adaptive search to extract minimal unsatisfiable subformula (MUS) of an unsatisfiable CNF instance. The hardness of clauses are approximately evaluated during the iterations of satisfiability testing. During each iteration, it increments the rank of clauses which are violated during the testing. The ranking system inherently represents the hardness of the clauses by the means of the heuristic. An adaptive core analyzes and adaptively changes such ranks until a IIS is found.

Bailey et al. [79, 80] also addressed the dual relation between IIS and the complement sets of maximal satisfiable subsets. When the complement sets are obtained approximately, the intersection between IIS and the complement sets is guaranteed.

Liffiton et al. [81, 76] proposed an approach to find the complement of all the *MSS* (Maximum Satisfiable Subformula). The *MSS* represents the maximal consistent subproblem which will become unsatisfiable by adding one supplement constraint on it. Since the IIS is the hitting sets of all complement sets, *CoMSS*, of the *MSS*, by enumerating all the *MSS*, the IIS can be identified. Their experimental results were not very impressive, as the computational time of finding all *CoMSS* of the problem is costly. It becomes even worse by including the hitting set resolution techniques.

Desrosiers et al. [82, 78] extend the methods proposed in [3] - *Removal*, *Insertion* and *HittingSet* on finding IIS in SAT and k-coloring instances. The number of Max-CSP solver calls for their approach is $\mathcal{O}(k)$, where k is the number of constraints inside an IIS. Alongside all these methods, the readers also can refer the survey in [83, 62] for other propositions in same approach.

2.2.3 Techniques analysis

The common component of these two approaches is to iteratively execute a solver. The significant difference between them is that the satisfiability approach iteratively executes an unsatisfiability testing solver, while the hitting set approach employs a Max-CSP solver to find assignments on CSP. Marques-Silva pointed out that the hitting set approach is less efficient than the satisfiability testing approach on SAT instances because the Max-SAT solver is less efficient when comparing their performance [62]. It is also addressed by Fu and Malik who construct a Max-SAT solver by employing *zChaff* [84] to iteratively identify and eliminate the MUC in SAT instances [85].

Also the experimental results reported in [82], demonstrates that the hitting set approach loses its performance on SAT instances. While the same authors shows the improvement in performance on identifying critical subgraph in k-coloring problem when their hitting set approach was applied. These observations demonstrate that:

- For the satisfiability testing approach: if the problem's satisfiability is relatively easy to prove, then adding/removing or the learning embedded strategy can successfully obtain an IIS.
- For the hitting set approach: if the problem's approximation solutions are relatively easy to obtain, then the iterative execution of approximated Max-CSP solver is more efficient than the satisfiability testing solver.

2.3 Motivation of this dissertation

In previous sections, the essential definitions of IS and IIS were introduced and followed by important properties and resolution techniques involving IS/IIS. The primary objective of this dissertation intends to adopt the concept and resolution techniques of IIS in telecommunication networks, furthermore, to identify the most interfered zone in a telecommunication network.

Since the number of the available frequencies is a limited resource for particular applications, before deploying a communication network, it is worth to estimate the interference among the antennae to verify if there is any interfered zone which causes service failure.

Such a problem can be modeled as the Frequency Assignment Problem (FAP), and its unsatisfiability/satisfiability is difficult to be determined due to the sizes of problems and the high complexity of the FAP problem itself [86]. *Satisfiability testing* may not be suitable for this application, since the unsatisfiability testing solver acts as the core engine during the search which will be executed a considerable number of times.

In paper [78], Desrosiers et al. adopted the methods proposed in [3] on detecting critical subgraphs in a graph. The critical subgraph can be considered as an IIS in the context of k-coloring problem. Thanks to the similarity between the k-coloring problem and the Frequency Assignment Problem, their approach may be considered as the ideal candidate for IIS identification in FAP.

The methods described in [78] are *Removal*, *Insertion* and *HittingSet*. Let H be the subset of critical constraints and C be the set of constraint of the problem. The *Removal* can be illustrated as in Algorithm 3. The algorithm iteratively removes the constraint which does not change the unsatisfiability on C. Such unsatisfiability is judged by a heuristic named *MinConflict*$()$. If by removing a constraint c, the constraints set C becomes satisfiable, the constraint c will be added in critical constraints subset H. The algorithm continues to remove the constraint in $C \setminus H$ and see if the unsatisfiability state of C will be changed or not. The procedure stops when *MinConflict*$()$ cannot find a partial consistent solution on critical constraints subset H.

The *MinConflict*$()$ is an approximate algorithm returning a subset of violated constraints C_v which are violated in an assignment found. Such an assignment respects the constraint satisfaction on H a priori, and minimizes the number of violated constraints in C. Regarding the similarity between k-coloring problem and the Frequency Assignment Problem (FAP) [86], the *MinConflict*$()$ can be implemented by a Tabu Search algorithm which is also suggested by Desrosiers et al. [78].

Algorithm 3: Removal [78]

Input : a set of constraints C
Output: a subset of constraints $H \subseteq C$ forming an IS

```
1  H ← ∅, C_v ← ∅;
2  repeat
3      choose a constraint c where c ∈ (C \ H);
4      C ← C − {c};
5      C_v ← MinConflict(C, H);
6      if C_v = ∅ then
7          H ← H ∪ {c};
8          C ← C ∪ {c};
9      end
10 until C_v ∩ H ≠ ∅ ;
11 return H;
```

Focusing on the *Min-Conflict* strategy, *TabuCol* adopts a very simple critical 1-move (or 1-exchange) [87] neighborhood structure. The neighbor solutions are generated by only changing the color (or frequency) of one node among all the conflict nodes.

Let $f(\mathcal{A})$ be the fitness function of a complete assignment $\mathcal{A}$, $\mathcal{A}(x_i)$ be the current value assigned on variable x_i in assignment $\mathcal{A}$ and (x_i, a_i) be a move by assigning the variable x_i with the new value a_i. *TabuCol* can be illustrated as in Algorithm 4.

Algorithm 4: *TabuCol*

Input : a graph $G = (V, E)$ and k number of colors
Parameters: $MaxIter$, L and λ
Output : a complete assignment $\mathcal{A}^*$ on G with k colors

```
1  Create a random assignment A;
2  set A* ← A and iter = 0;
3  Set TabuList ← ∅;
4  repeat
5      iter ← iter + 1;
6      Choose a candidate 1-move (x_i, a_i) among all conflict variables with minimum violation;
7      Add move (x_i, A(x_i)) into TabuList for L + λ × Conflict(A) iterations;
8      Set A ← A + (x_i, a_i);
9      if f(A) < f(A*) then A* ← A
10 until f(s) = 0 or iter = MaxIter ;
```

The fitness function $f()$ in line 9 measures the number of violated arcs in the current assignment $\mathcal{A}$. Thanks to the similarity between k-coloring problem and FAP, the neighborhood structure defined in *TabuCol* is suitable for the case of FAP which changes the frequency value on one variable in conflict.

Based on its simple 1-move neighborhood structure, the algorithm exploits a relatively small part of the search space. Such a strategy dramatically minimizes the runtime while at the same time, the promise on performance is kept.

In addition to the *Removal* algorithm, the same authors of [3] proposed another algorithm called *Insertion*. The algorithm can be expressed as in Algorithm 5.

Algorithm 5: Insertion [3]

Input : a set of constraints C
Output: a subset of constraints $H \subseteq C$ forming an IS

```
1  H ← ∅, C_v ← ∅;
2  repeat
3      if C_v ≠ ∅ then
4          C ← C \ C_v;
5          choose one constraint c ∈ C_v;
6          H ← H ∪ {c};
7      end
8      C_v ← MinConflict(C, H);
9      if C_v = ∅ then return Fail to find IIS;
10 until H ∩ C_v ≠ ∅ ;
```

In Algorithm 5, *MinConflict*() is the same approximated algorithm mentioned in Algorithm 3. The algorithm stops when *MinConflict*() cannot find a solution respecting all constraints in H. The constraint set H increases its size at each iteration by choosing a constraint c from the violated constraint set of *MinConflict*(). As in *Removal*, line 9 stops the algorithm if it fails to find an IIS.

The primary difference between these two algorithms is that *Removal* removes the constraint one by one from C per iteration, while the latter removes a set of constraints. In order to locate the first critical constraint in H, *Removal* may involve a significant amount of iterations while *Insertion* processes only one iteration. The results reported in the paper of Desrosiers et al. [78] prove the efficiency of *Insertion* compared to *Removal* on the k-coloring problem. After a first constraint c is identified and added into the constraint set H, both algorithms will exclusively construct an IIS on this initial constraint c.

Our experiments on the performance of *Insertion* are conducted on the CELAR benchmark (see Section 3.2.1 for detail information). Table 2.1 demonstrates the results obtained on the over-constrained instances in CELAR. The second and third columns indicate the sizes of variables and constraints of the IIS found by *Insertion*. The final column shows the number of successful runs in 10 executions, the N/A means that the unsatisfiability of subproblem identified by *Insertion* cannot be proven in 30 minutes.

Analyzing the cause of the exponential verification time, we observe that the size of the subproblem identified by *Insertion* is too big to verify its unsatisfiability in a reasonable time. It raises the need of a new algorithm to deal with

CELAR	IIS variables size	IIS constraints size	Successful rate
Scen04	5	10	2/10
Scen05	5	10	2/10
Scen06	N/A	N/A	N/A
Scen07	9	26	2/10
Scen08	N/A	N/A	N/A
Scen09	7	19	1/10

Table 2.1: Results of *Insertion* on CELAR benchmark

the frequency assignment problem to find smaller IIS if it exists, or same size IIS within less time. In the next chapter, a new hybrid algorithm will be proposed to improve IIS identification on FAP instances.

2.4 Conclusion

In this chapter, the main topic of this dissertation, IIS identification, is introduced. It begins with the basic definitions and properties. There are two definitions based on different views, one is based on the key feature of *Irreducible*, the other is derived from the *Minimal* form of IIS. The *Irreducible* indicates the state change between unsatisfiability and satisfiability, the IIS becomes satisfiable if any one of its variable/constraint is removed. *Minimal* means there is no strict inclusion between two IIS, that is the two IIS may intersect, but one IIS cannot contain another IIS.

Following basic definitions, an essential survey of algorithms in the literature is presented under two different approaches can be identified. The satisfiability testing approach features a satisfiability testing solver, which iteratively proves the unsatisfiability or satisfiability by removing (inserting, respectively) variable (constraint, respectively) from the testing subproblem. The hitting set approach consists in iteratively executing a Max-CSP resolution algorithm to generate a collection of violated constraint subsets. Thus the IIS can be identified by finding a hitting set of a collection of constraint subsets. The theoretical knowledge behind this approach is that the subset of constraints belonging to an IIS has an intersection with the violated constraints subsets found by Max-CSP algorithm.

We summarize these approaches, and conclude that the unsatisfiability of the instance is relatively easier to be proven, then the satisfiability testing approach is suitable to solve such instance. When the maximal satisfaction of constraints can be achieved conveniently, the hitting set approach is a right way.

At the end of this chapter, we conduct the examination of algorithm performance on one Frequency Assignment Problem benchmark, CELAR. The candidate algorithm is *Insertion* algorithm proposed by Galinier and Hertz [3]. The experimental result carried out demonstrates the low-effectiveness on CELAR instances. The algorithm fails to determine the small IIS and is not robust on CELAR instances.

Based on our analysis, we intend to propose a new approach which will achieve better performance than the existing approach on telecommunication applications. The new approach is requested to be fast and robust on IIS detection. The effectiveness and efficiency will be critical features in the new method.

In next chapter, we will concentrate on adopting existing resolution techniques on the applications of Frequency Assignment Problem and k-coloring problem.

Chapter 3

IIS in frequency planning

The growing demands in radio communication networks and the spectrum rarity these days make radio frequencies more precious than ever. Due to the frequency scarcity, the number of frequencies available is not sufficient to deliver enough capacity to all systems and applications. The only solution to this problem is to maximize the frequency reuse between them, but this strategy may generate interference zones or service perturbation zones inside the telecommunication network, which is the nightmare for every network operator either for civil or for military usage. When it is impossible to reuse the frequency without interference, it is still possible to ease the frequency planning and to reduce interference by changing the network design or parameter settings. Then the question is ”which are those interference constraints causing the service perturbation zone in the Frequency Assignment Problem (FAP), and how can we avoid the creation of a perturbation zone at the network design step?”. If such constraints can be exactly detected then the decision will be to change the parameter settings of the network transmitters involved in these constraints.

The main objective of this chapter is to propose an approach which is able to identify these infeasible interfering constraints in a telecommunication network in the context of military applications. This approach delivers an effective and efficient decision tool for the network operators to identify the service perturbation zone in the telecommunication network which cannot be solved during the deployment step. Based on the CSP formulation of FAP, such zone can be considered as an *Irreducible Infeasible Subset* which is over-constrained by the electromagnetic constraints issued from the network parameters settings. So this chapter is devoted to the research on identifying the perturbation zone in one network, an analogous to the IIS in the telecommunication application. This problem is quite new for FAP and was firstly tackled by [61] as we will see in the next section.

The sections of this chapter will be organized as follows. In Section 3.1, the global properties of FAP and our specific FAP are defined. The FAP benchmarks we used to evaluate our proposals are illustrated in Section 3.2 with their topology and properties. Then our work is presented in two steps corresponding to two different algorithmic approaches we developed. Section 3.3 will address the first algorithm which is directly derived from the method

proposed in the literature. After that, by analyzing the advantage and drawback of this approach, a second and new algorithm to detect an IIS in the FAP is presented in Section 3.4. The experimental results of the new method on the FAP benchmarks will be demonstrated in Section 3.6. Finally, a general conclusion of this chapter will be given in Section 3.7.

3.1 Introduction

Respecting physical law, the radio bandwidth and the number of frequencies are precious resources which are limited for each communication system. The availability of radio spectrum is regulated by the International Telecommunication Union (ITU) at a world-wide level for each system and by the governments at nation level for each operational usage of the system in the country. The operators are freely or economically licensed to use one or several frequency bands to deploy services. The frequency band is mostly represented by $[f_{min}, f_{max}]$ in which f_{min} is the lowest available frequency and f_{max} is the highest available frequency in the spectrum. As example, the Figure 3.1 shows the frequency intervals of spectrum regarding the usages in USA up to 1GHz. Regarding the growing of radiocommunication demands, the efficient usage of radio spectrum is an important problem on which research teams are hugely involved since the 1990's.

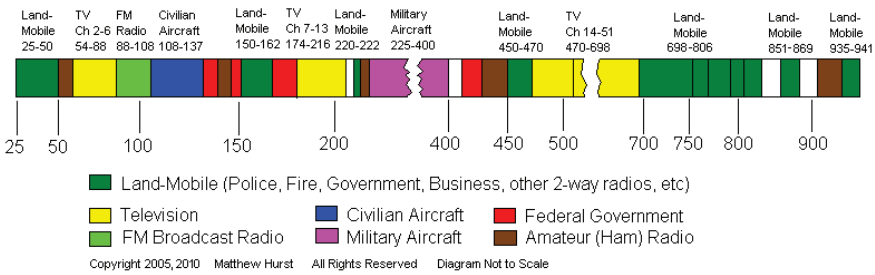

Fig. 3.1: Radio spectrum for wireless communication in USA
(Copyright©radio-scanner-guide.com)

Within a given system, when the number of required communications exceeds the available radio channel resources, the frequencies are reused on several transmitters to deliver additional radio channel resources. Unfortunately the interference generated by the different signals using the same channel or adjacent channels may globally decrease the service quality and the expected network capacity. In such conditions frequency reuse creates service perturbation zones in the network which correspond to zones with radio interference above a given threshold for each service. The crucial question is whether it is possible to identify the existence of such zones before the network deployment

to eventually modify the network parameters settings and avoid interference. Such a question may be answered by modeling and solving the Frequency Assignment Problem (FAP).

The FAP [86] consists in assigning frequencies to the transmitters of a radio telecommunication network. The aim is to benefit from the geographical separation of assigned frequencies to reuse the radio spectrum efficiently and to minimize interference in the network. There are many version of the FAP, as many versions as the number of radio systems, a summary is given in [88]. The version of the problem considered here is the fixed FAP whose assignment of frequencies on transmitters are not changing with time, as opposed to the dynamic FAP where the available spectrum, the number of frequency per transmitters and/or the frequency index may change [89]. In the fixed FAP the available spectrum is initially defined and usually consists of a set of consecutive equally spaced channels, possibly with some gaps of one or more channels which are unavailable for subsets of transmitters. This version of the FAP is still important in a wide range of terrestrial and satellite based radio systems, both for civil and military applications.

The particular case of the Radio Link Frequency Assignment Problem (RLFAP) considers the network as a set of radio stations equipped with antennae and the radio link among them. The station is both a transmitter and a receiver and the interference arises on the receiving signals. The network can be illustrated by a directed constraint graph, in which the stations are the nodes and the radio links are the directed arcs. The link (i, j) expresses a one way of transmission from the station i to the station j. In RLFAP the frequencies are assigned to the radio links, while in some other problems, like the FAP in cellular networks, the frequencies are assigned to the stations. In our RLFAP each radio link requires only one frequency to carry the communications.

The radio link quality is measured at the receiver by the Signal-to-Interference-plus-Noise Ratio (SINR), or interference ratio, and several interfering signals may be received in the same time in addition to noise. In the case of single interfering signal the SINR involves only two signals which are the carrier link and the single interfering link, then we may express the SINR by a binary interference constraint between these links. Most of time, the constraint will define a spectrum separation to respect between both frequencies assigned to the links; the separation value will depend on the geographical proximity between the links or the difference of received power on each link. If there are several interfering links, the constraint will be n-ary and involves simultaneously the carrier and all interfering links. In that case, there is no specific frequency separation to respect but we define a threshold for the SINR and the frequency assignment on all links will have to satisfy the global threshold. These constraints are very difficult to satisfy as they involve more than two variables. Figure 3.2 shows one station with three different received signals, each one will be successively the carrier and the other both the interfering signals for the selected carrier, the 3-ary constraint will have to be satisfied for the 3 cases.

Considering the constraints to satisfy, the objectives of RLFAP include to minimize the number of frequencies used, to minimize the span between the lower and the higher frequency used in the spectrum, to minimize the number of unsatisfied constraints for a given set of frequency, etc. There are many options as described in [90]. The main theoretical model to represent this set of problems is the class of graph coloring problems such as graph coloring, k-graph coloring for k colors, T-coloring where T defines the forbidden channel separations, set-coloring problems where a set of frequencies may be assigned to the variables, etc., where sometimes the graph is an hypergraph. References on these problems can be found in [91].

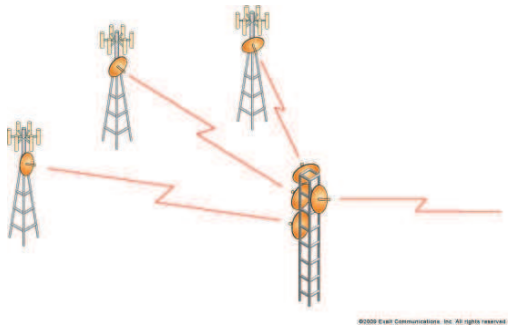

Fig. 3.2: RLFAP
(Copyright©Exalt Comm. Inc,.)

In addition to frequency assignment, the wave plane polarization also plays an important role in interference management. The plane polarization is a property of radio waves that describes the orientation of the wave plan, and can be horizontal or vertical. In such a case, the wave plane assignment is an extension of the standard FAP, the separation between frequencies or the SINR threshold are various according to the polarity of the carrier and the interfering signals. In practice, the constraints are easier to respect, i.e. a smaller frequency separation or lower SINR threshold, if the polarities are different. Assigning simultaneously the polarity and the frequency brings more complexity to the problem because we have more variables and the constraints difficultly depend on polarity choice.

Based on the nature of the problem characteristics, the RLFAP can be formulated as a CSP. It consists of a finite set of available frequencies, the CSP domain, a finite set of radio links, the CSP variables, and a finite set of interference constraints, the CSP constraints respectively. If the problem is feasible, the aim of RLFAP consists in assigning a frequency to all radio links while satisfying the constraints and minimizing one of the standard FAP objective mentioned above; if the problem is infeasible the aim is to minimize the number of unsatisfied constraints, and hence the residual interference.

In order to solve the RLFAP problem as a CSP and evaluate the performance of the methods, many benchmarks already described as CSP have been proposed by the telecommunication or operational research community mainly

for military systems. Table 3.1 shows the characteristics of three sets of instances frequently used: CELAR, GRAPH, ROADEF2001 and the original problem SOES. The rows Unary, Binary and N-ary indicate the categories of constraints involve in the benchmark. The row polarization indicates the benchmarks using polarity and frequency assignment all together. The line data source indicates the simplified real world instances, the academic randomly generated instances and the modified real world instances respectively.

The CELAR benchmark consists of 11 instances, the problem size is from 200 to 916 variables, with 1235 to 5744 constraints and the maximal number of available frequencies is 48. The GRAPH benchmark contains 14 instances with 200 to 916 variables, 1134 to 5246 constraints and a maximum of 48 frequencies. CELAR and GRAPH instances involve binary constraints exclusively.

The ROADEF2001 benchmark contains 40 instances with 200 to 3000 variables and 1108 to 41781 constraints. The maximal number of frequencies is 998 which includes the two polarities. The density of the instances is higher than the ones in CELAR and GRAPH. Also ROADEF2001 only contains binary constraints with or without polarity. All information about these benchmarks can be collected on the well known web page dedicated to FAP (see http://fap.zib.de/problems/).

The SOES benchmark is the most recent and contain more problem features. SOES has 20 instances with 16 to 2000 variables and 73 to 13669 constraints. The numbers of available frequencies is from 49 to 727.

Benchmark	CELAR	GRAPH	ROADEF2001	SOES
Nb. instances	11	11	40	20
Nb. variables	200-916	200-916	200-3000	16-2000
Nb. constraints	1235-5744	1134-5246	1108-41781	73-13439
Nb. Freq	48	48	182-998	49-727
Density	0.01-0.07	0.01-0.06	0.01-0.6	0.006-1.22
Unary				x
Binary	x	x	x	x
Polarization			x	x
N-ary				x
Data source	simplified	academic	modified	modified

Table 3.1: Four Benchmarks of RLFAP

Comparing the four sets of instances, the number of variables and constraints, and the density, are relatively smaller for CELAR and GRAPH. However the density is not sufficient to explain the structures of these benchmarks. With the aid of graph visualization the differences among the benchmarks can be shown more clearly. The Figures 3.3 to 3.6 illustrate the difference in graph topology between these benchmarks on 4 particular instances. The topologies of GRAPH, ROADEF2001 and SOES are quite similar but very different from CELAR which is really simplified. Many CELAR instances are composed of branches with different densities in which there are several very dense

subproblems. The visual graphs of GRAPH, ROADEF2001 and SOES instances look like nests, the nodes are strongly connected with each other.

From these figures, we indicate for each instance one Infeasible Subset with bold lines. In higher density graphs ROADEF2001 and SOES, the Infeasible Subsets are difficult to see due to the graph density. In the next section, we present more details and a deeper analysis of each benchmark.

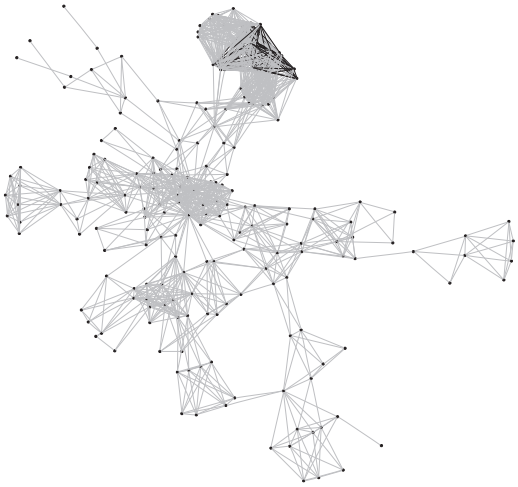

Fig. 3.3: Topology scen02 in CELAR

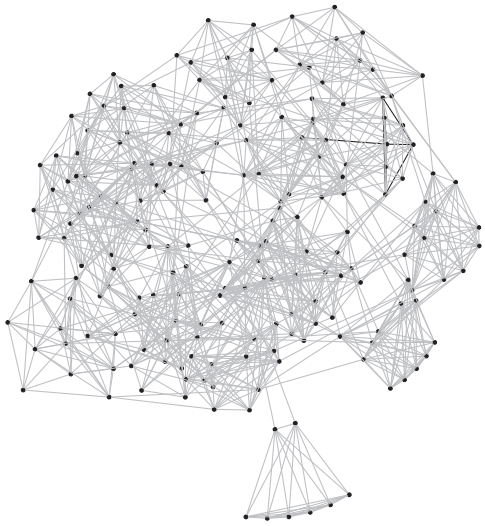

Fig. 3.4: Topology graph03 in GRAPH

3.2 Benchmark description

The CELAR/GRAPH and ROADEF2001 are publicly available for all researchers. The reason for choosing CELAR/GRAPH and ROADEF2001 for our work is that these benchmarks are widely adopted by the artificial intelligence and operational research communities to evaluate their algorithms. It provides an open platform to compare a new approach with the existing approaches. The SOES benchmark is available privately during our research period. It is the evaluation benchmark which allows us to compare the performance between DGA's (Délégation Générale pour l'Armement) previous approach on Irreducible Infeasible Subset and our new proposal. In following sections, these four benchmarks will be illustrated in detail.

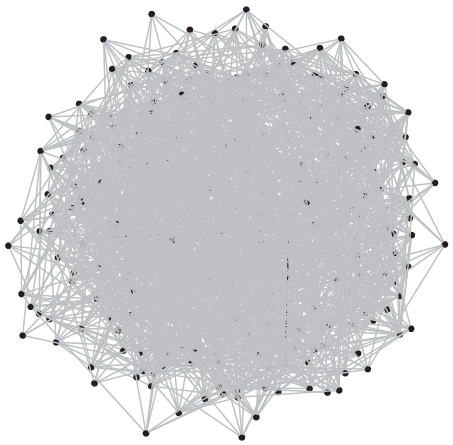

Fig. 3.5: Topology fapp16 in ROADEF2001

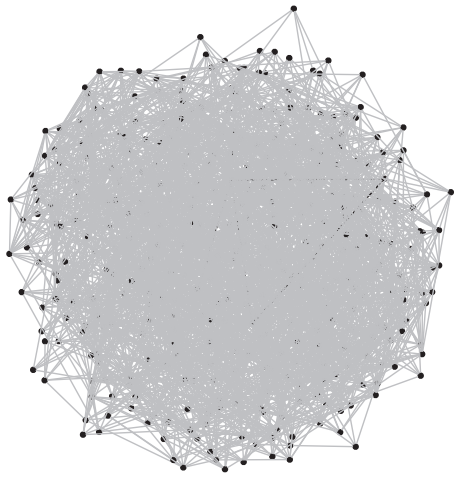

Fig. 3.6: Topology scen20 in SOES

3.2.1 Radio Link Frequency Assignment Problem - CELAR and GRAPH

The CELAR benchmark is a set of RLFAP instances defined in the framework of the European project EUCLID CALMA (Combinatorial Algorithms for Military Applications) [92]. All problem instances have been built from an unique real-life instance with 916 links and 5744 constraints (hard and soft).

The GRAPH instances (Generating Radio link frequency Assignment Problems Heuristically) have been proposed by a group at Delft University of Technology. These instances are randomly generated by van Benthem [93], and hold the same characteristics as CELAR instances. The GRAPH instances are similar to CELAR's by their problem structure and their hardness but differ by further randomized constraints. Taking into account preliminary experimentation, the author of GRAPH conjectures that instances generated by GRAPH are generally slightly harder to solve than the CELAR ones.

The aim of both RLFAP benchmarks consist in assigning a limited number of frequencies to a set of radio links defined between pairs of sites in order to minimize the number of frequencies used and the highest frequency used if the problem is feasible, or to minimize a weighted sum of violated constraints if the problem is unfeasible [92]. The RLFAP is known to be very hard to solve, due to its close relation to the vertex coloring problem. Koster et al. [94] prove (using a reduction from maximum satisfiability) that RLFAP is NP-hard. Each radio link is represented by a variable whose domain is the set of all frequencies that are available for this link. Most constraints involve two variables f_i and f_j such that:

$$|f_i - f_j| > \delta_{ij} \tag{3.1}$$

The two variables f_i and f_j represent two radio links with interference. The constant δ_{ij} defines the spectrum separation available for these links to avoid the interference, its value depends on the amount of interference between them obtained by computing the SINR. In addition to these constraints there is an equality constraint for duplex links between any couple of stations, the separation of frequency for duplex links must be exactly equal to 238.

Table 3.2 shows the characteristics of CELAR and GRAPH instances where $Nb.Var$ is the number of variables, $Nb.Ctr$ is the number of constraints, $HighestFreq$ is the highest frequency used for each domain, AC column indicates whether the instance is Arc-Consistent or not (C or INC) and density is the graph density. For our experiments, the instances with the objective of minimum span have been converted to infeasible instances by removing several upper frequencies from the domains, and the instance with minimum order are pruned from the benchmarks so we only keep 9 GRAPH instances.

CELAR	Nb. Var	Nb. Ctr	HighestFreq	AC	Density
scen01	916	5548	666	C	0.01
scen02	200	1235	380	C	0.06
scen03	400	2760	380	C	0.03
scen04	680	3967	666	INC	0.02
scen05	400	2598	666	INC	0.03
scen06	200	1322	792	INC	0.07
scen07	400	2865	792	INC	0.04
scen08	916	5744	792	INC	0.01
scen09	680	4103	792	INC	0.02
scen11	680	4103	666	C	0.02
GRAPH	**Nb. Var**	**Nb. Ctr**	**HighestFreq**	**AC**	**Density**
graph03	200	1134	366	INC	0.06
graph04	400	2244	380	INC	0.03
graph05	200	1134	792	INC	0.06
graph06	400	2170	792	INC	0.03
graph07	400	2170	792	INC	0.03
graph10	680	3907	380	INC	0.02
graph11	680	3757	792	INC	0.02
graph12	380	4017	792	INC	0.06
graph13	916	5273	792	INC	0.01

Table 3.2: Characteristics of CELAR and GRAPH

3.2.2 RLFAP with polarization - ROADEF2001

In the Frequency Assignment Problem with Polarization (FAPP), each radio link is assigned a frequency polarization pair (f_i, p_i), where f_i is the frequency on the transmitter i and p_i is the polarity. The components of the pair represent the frequency carrying the transmitted signal and its wave polarization. Two signs +/- indicate the directions of the polarization, which are the horizontal and vertical directions respectively. In the ROADEF2001 benchmark, the constraints can be divided roughly into two categories - Imperative Constraints (IC) and Electromagnetic interference Constraints (EMC) [95]. The IC constraints are regarded as constraints which should be absolutely satisfied. The IC constraints can be classified into:

- Frequency equality or inequality constraint: $f_i = f_j$ or $f_i \neq f_j$,
- Interval equality or inequality constraint: $|f_i - f_j| = \varepsilon_{ij}$ or $|f_i - f_j| \neq \varepsilon_{ij}$,
- Polarity equality or inequality constraint: $p_i = p_j$ or $p_i \neq p_j$.

Where ε_{ij} is the authorized or non-authorized distance between two frequencies. In contrast to IC constraints, the EMC constraints are considered as soft constraints which can be violated at a certain level. In the context of ROADEF2001 challenge, the instances have 11 levels identified by an integer index k within a range $[0, 10[$. The level 0 indicates the strictest condition, and the level 10 means the most relaxed level.

$$|f_i - f_j| \geq \begin{cases} \gamma_{ij}^0 \geq \gamma_{ij}^1 \geq \cdots \geq \gamma_{ij}^{10} \text{ , if } p_i = p_j \\ \delta_{ij}^0 \geq \delta_{ij}^1 \geq \cdots \geq \delta_{ij}^{10} \text{ , if } p_i \neq p_j \end{cases} \tag{3.2}$$

By introducing the polarization, each frequency has an option between two polarities, consequently, it increases the complexity. Usually in the case of equality of polarities, the distance between two frequencies γ_{ij}^k defined by an EMC constraint is larger than the case of the inequality distance δ_{ij}^k at the same level k. Two frequencies f_i and f_j may not satisfy on an EMC constraint in the same direction of polarity, but may satisfy the condition in the case of a polarities inequality. Based on Equation 3.2, if two frequencies polarization pairs (f_i, p_i) and (f_j, p_j) satisfy level k, they also satisfy all relaxed levels from k, that is all levels greater than k. For example, if two pairs satisfy the level 7, they strictly satisfy all the levels greater than 7, which are $8, 9$ and 10.

An assignment of the FAPP problem consists of a set of the frequency and polarization pairs selected from the available sets of frequencies and polarities of all radio links. An assignment is called a solution S, if it satisfies all the imperative constraints. A solution S is said to be k-feasible, if S satisfies all the imperative constraints and all EMC constraints at level k and all relaxed levels from k. A trivial level 11 is introduced in case that there is no consistent solution at level 10. If there is no feasible solution at level 0 for an instance, ROADEF2001 defines three objectives

in lexicographic order which are firstly to search for k^*, the smallest relaxation level for which a k^*-feasible solution exists, secondly to minimize the number of EMC constraints not satisfied at level $k^* - 1$ and lastly to minimize the number of EMC constraints not satisfied at all levels lower than $k^* - 1$.

The number of permutations for FAPP problem is $\prod_i^n |d_i| \times |p_i|$, in which n is the number of transmitters, $|d_i|$ is the number of available frequencies for the transmitter i and $|p_i|$ is the number of polarities for the transmitter i. As the k-coloring problem, the FAPP problem itself is NP-hard [86].

In Table 3.3, the basic characteristics of ROADEF2001 instances are presented. the columns 2,3 and 4 indicate the size of instances with the numbers of variables, the numbers of constraints and the numbers of frequencies. The AC level gives the highest level of Arc-Consistency for each instance, and Feasible level indicates the highest level of instance for its feasibility (k feasible).

scen	Nb. Var	Nb. Ctr	Nb. Val	AC level	Fea. level	scen	Nb. Var	Nb. Ctr	Nb. Val	AC level	Fea. level
fapp01	200	1108	190	**3**	**4**	fapp21	500	1589	242	4	4
fapp02	250	1636	210	2	2	fapp22	1750	16924	802	7	7
fapp03	300	2327	250	7	7	fapp23	1800	33337	302	9	9
fapp04	300	1799	270	1	1	fapp24	2000	14301	302	7	7
fapp05	350	2488	270	**8**	**11**	fapp25	2230	11974	302	3	3
fapp06	500	3478	290	5	5	fapp26	2300	12761	302	7	7
fapp07	600	4778	302	9	9	fapp27	2550	6231	242	5	5
fapp08	700	3834	282	5	5	fapp28	2800	12046	998	3	3
fapp09	800	4800	350	3	3	fapp29	2900	41781	998	6	6
fapp10	900	6071	362	6	6	fapp30	3000	33301	778	7	7
fapp11	1000	8005	362	8	8	fapp31	400	1644	700	**3**	**5**
fapp12	1500	13439	310	2	2	fapp32	550	5017	998	6	6
fapp13	2000	13669	190	3	3	fapp33	650	4631	498	5	5
fapp14	2500	21610	362	4	4	fapp34	750	4623	998	4	4
fapp15	3000	17754	182	5	5	fapp35	1500	11723	698	6	6
fapp16	260	2088	302	11	11	fapp36	2000	10067	454	7	7
fapp17	300	2056	302	4	4	fapp37	2250	22553	998	5	5
fapp18	350	2387	302	8	8	fapp38	2500	32622	698	3	3
fapp19	350	3114	802	6	6	fapp39	2750	12605	502	**2**	**3**
fapp20	420	2487	302	10	10	fapp40	3000	28313	698	4	4

Table 3.3: ROADEF2001

3.2.3 RLFAP with polarization and n-ary constraints - SOES

SOES is the latest RLFAP benchmark proposed by the CELAR in 2008. These instances were delivered in the context of a DGA (Délégation Générale pour l'Armement) project in which we were involved in a consortium (THALES, SILICOM, ONERA, UTBM). The instances are based on real world problems modified to assess the performance of operational research methods. The main novelty of this benchmark is that in addition to well-known unary and binary RLFAP constraints, SOES contains also n-ary constraints. Polarization assignment is also included in the benchmark in a different way from ROADEF2001 as the different polarities extend binary constraints in the form of:

$$|f_i - f_j| \in \begin{cases} \Delta_{ij}^{++}, & \text{if the polarities of } f_i \text{ and } f_j \text{ are both vertical;} \\ \Delta_{ij}^{+-}, & \text{the frequency } f_i \text{ with vertical polarization, and the other with horizontal one;} \\ \Delta_{ij}^{--}, & \text{if the polarities of } f_i \text{ and } f_j \text{ are both horizontal;} \\ \Delta_{ij}^{-+}, & \text{the frequency } f_i \text{ with horizontal polarization, and the other with vertical one;} \end{cases} \tag{3.3}$$

If we only consider the equality and inequality of frequency polarities, the binary constraints are formulated as:

$$|f_i - f_j| \in \begin{cases} \Delta_{ij}^{=}, & \text{if equality of polarization;} \\ \Delta_{ij}^{\neq}, & \text{if inequality of polarization;} \end{cases} \tag{3.4}$$

Equations Equ 3.3 and Equ 3.4 describe the binary constraints with polarization in the SOES instances, accompanying with the non-polarization variant (Equ. 3.5).

$$|f_i - f_j| \in \Delta_{ij} \tag{3.5}$$

There are two types of n-ary constraints which are: the perturbation constraint (Equ. 3.6) and the intermodulation constraint (Equ. 3.7),

$$\sum_i \beta_{ip} T_{ip}(|f_i - f_p|) \leqslant B_p \tag{3.6}$$

$$|f_p - (\pm\alpha_i f_i \pm \alpha_j f_j)| \geqslant \xi \tag{3.7}$$

where f_p is the carrier frequency and f_i and f_j are used by the interfering signals. These constraints correspond to a multiple interference case where the interfering signals generate a global interfering signal on the carrier. For perturbation constraints, the threshold B_p indicates the expected reception quality, β_{ip} depends on the budget link between the carrier and the interferer and T_{ip} depends on the frequency separation between the carrier and the interferer. For intermodulation constraints, the α coefficients generate intermodulation interference arising from two initial interfering frequencies. In this thesis, our objective in this benchmark is to find an Irreducible Infeasible Subset based on the binary constraints exclusively, thus the n-ary constraints will not be used.

The sizes of SOES instances are shown in Table 3.4, where $Nb.Fq$ is the number of available frequencies, $Vars$ is the number of variables and $Ctrs$ is the number of constraints. The sizes of instances of SOES are largely different from the ones in ROADEF2001 which are relatively smaller.

Scenario	Nb. Fq	Nb. Vars	Nb. Ctrs	Scenario	Nb. Fq	Nb. Vars	Nb. Ctrs
SCEN 01	91	50	172	SCEN 11	100	770	4041
SCEN 02	123	47	185	SCEN 12	72	702	3623
SCEN 03	100	40	166	SCEN 13	700	182	3297
SCEN 04	143	16	73	SCEN 14	91	300	2593
SCEN 05	99	50	465	SCEN 15	155	1500	13439
SCEN 06	75	38	859	SCEN 16	727	121	840
SCEN 07	49	50	187	SCEN 17	71	568	4140
SCEN 08	49	50	197	SCEN 18	95	2000	13669
SCEN 09	50	48	229	SCEN 19	60	154	2928
SCEN 10	49	50	208	SCEN 20	577	249	2938

Table 3.4: SOES instances

3.3 A 2-phase algorithm to identify IIS

To best of our knowledge, the algorithm *wcore* proposed by Hemery et al. [61] is the first algorithm to detect an IIS on CELAR and GRAPH instances. *wcore* adopts the satisfiability testing approach mentioned before, which embeds a variable selection heuristic inside an exact satisfiability solver. The variables selection heuristic chooses the *critical* variables during the *MAC* search and moves them toward the top of the search tree. The critical variables are measured by the weights generated by a variable selection heuristic. An IIS is detected by proving the unsatisfiability on the subset of selected variables.

This approach is very effective if the infeasibility of the instance is easy to prove. Thanks to the problem specific techniques (unit propagation, clause learning, resolution graph) in SAT, the infeasibility of a SAT instance is relatively

easier to be proven [62]. Unfortunately it is not always the case for RLFAP and FAPP, due to the complexity of the instances.

In the previous chapter, we presented two algorithms as the candidates for identifying IIS in RLFAP and FAPP, namely *Removal* and *Insertion*. Both algorithms are proposed by Galinier and Hertz [3] and adopted by Desrosiers et al. [78] in detecting a critical subgraph of the k-coloring problem. They consist in iteratively executing an embedded heuristic and identifying the critical constraints potentially belonging to an IIS. The embedded heuristic *MinConflict*() is dedicated to minimize the sum of the violated constraints during the search.

The *Removal* algorithm removes the constraints one by one until the rest of the problem becomes feasible. The last removed constraint causing the transition between the unsatisfiability to the satisfiability is recorded and put again into the rest of problem as a critical constraint. During the iteration, *MinConflict*() will satisfy all the critical constraints a priori by weighting them in its objective function. *Removal* stops when it fails to find a partial solution on all critical constraints.

In contrast to *Removal*, *Insertion* considers one violated constraint as a critical constraint among all violated constraints found per iteration of *MinConflict*(). The rest of the violated constraints is directly pruned from the problem. *MinConflict*() works in the same manner as in *Removal*. The procedure of *Insertion* stops when there is no partial solution on all critical constraints. Theoretically, if *MinConflict*() fails to violate exactly one constraint per IIS, *Insertion* has the risk to destroy the IIS by removing one more constraint from it.

Beside the *Removal* and *Insertion* algorithms, in the paper of Desrosiers et al. [78], the authors proposed another algorithm called *prefiltering*. The algorithm can be illustrated as:

Algorithm 6: prefiltering [3]

input : a set of constraints C
output: a subset of constraints $H \subseteq C$ forming an IS

```
1 H ← ∅, C_v ← ∅;
2 repeat
3     if C_v ≠ ∅ then
4         H ← H ∪ {C_v};
5         C_v ← ∅;
6     end
7     C_v ← MinConflict(C, H);
8 until H ∩ C_v ≠ ∅ ;
```

It is originally developed as a pre-procedure to reduce the size of instances. Comparing the pseudo-code of *Insertion* and *prefiltering* algorithms, *prefiltering* puts all violated constraints C_v into the critical constraints set H and never prunes the constraints from the whole constraint set C. With such a difference, *prefiltering* highlights all violated

constraints found by *MinConflict*() and leverages their importance by converting them into the critical constraints set H. By preserving all violated constraints per iteration, there is no risk to prune any constraints belonging to an IIS from the problem. Thanks to such a feature, it has no risk to prune more constraints from IIS even with relatively poor performance of *MinConflict*() on FAP instances.

Algorithm 7: 2-phase algorithm

input : a set of constraints C
output: a subset of constraints $H \subseteq C$ forms an IS
$H \leftarrow \emptyset, C_v \leftarrow \emptyset, W_{C_v} \leftarrow \emptyset$;
Given an initial solution on C;
`/* Construction phase */`
repeat
  if $C_v \neq \emptyset$ **then**
    $c \leftarrow Select(C_v, H, W_{C_v})$;
    $H \leftarrow H \cup \{c\}$;
    $H \leftarrow Saturate(H, C)$;
  end
  $[W_{C_v}, C_v] \leftarrow MinConflict(C, H)$;
until $H \cap C_v \neq \emptyset$;
`/* Verification phase */`
if $MAC(H)$ *infeasible* **then**
  return *subset constraints* H*;*
end

We adopt the *prefiltering* algorithm and propose a 2-phase method to detect the IIS in CELAR instances [96]. The algorithm (Algorithm 7) is composed by two phases - the construction phase and the verification phase. The construction phase will be mainly based on *prefiltering* algorithm which iteratively selects the violated constraint in C_v and adds it into the critical constraint set H. Weights on violated constraints W_{C_v} will be generated by *MinConflict*() as the indicator of the satisfaction difficulty [47]. The verification phase will consist in verifying the infeasibility/feasibility of the constraints set H by an exact algorithm *MAC*(). Detailed information of these two phases will be presented in following sections.

3.3.1 Construction phase

In the construction phase, *MinConflict*() stops when it cannot find a partial solution on the critical constraints set H. It attempts to satisfy the constraints of H by introducing heavy weights on these constraints into its fitness function. At same time, it minimizes the violation on the rest of constraints, denoted $C \setminus H$. The critical constraints set H is

constructed iteratively by choosing one constraint per iteration from C_v which is the violated constraints subset found by *MinConflict*(). The chosen constraint has the heavier weight in W_{C_v}.

It is noticed that *MinConflict*() will be executed numerous times. Since only one constraint will be identified per iteration, the number of executions of *MinConflict*() equals the number of critical constraints. It is evident that the *MinConflict*() cannot be a costly procedure, otherwise the computational performance cannot be guaranteed.

In order to keep high performance on IIS identification, *TabuCol* [97] is chosen as a perfect candidate playing the role of *MinConflict*(). *TabuCol* employs a *critical 1-move* neighborhood structure. In the FAP, we choose the same strategy and only accept the move on the variables of violated constraints.

A slight difference from original *TabuCol* is that the *MinConflict*() need to respect the satisfaction on the constraints in subset H. It appears that a dedicated objective function is needed for such purpose. The objective function needs to distinguish the importance of critical constraints in H from constraints in $C \setminus H$.

$$f = \alpha \times (|H|) + |C_v| \tag{3.8}$$

In Equation 3.8, $|H|$ is the number of violated critical constraints, $|C_v|$ is the number of the violated constraints in C except those in H, and α is a sufficient large constant which distinguishes the constraints in H from the constraints in C_v. The value of α is set as the number of greatest node degree of the testing instance during the experiments, which equals the maximal number of constraints involved on the same variable.

MinConflict() searches an assignment which has minimal cost on the above objective function and returns a violated constraints subset C_v. The weights W_{C_v} are generated by *MinConflict*() on all violated constraints C_v during the search. The weights on violated constraints W_{C_v} are carried out by measuring the difficulty of satisfaction during search. The weight on each constraint will be initialized with zero and increased one unit in case that it is violated during each iteration of *Tabu*. *MinConflict*() stops when the cost of the fitness function cannot be improved during a given number of iterations. In our experiments, the maximal iteration is set to 1000.

Based on these weights, *Select*() chooses one violated constraint from C_v with the most weight in W_{C_v}. Even though is no direct evidence that the weighting system can identify correctly the constraints belonging to an IIS, numerous studies show that the weighting system can guide the search toward the harder subproblem which may be potentially an IIS [47], or indicate the subset of critical constraints which are hard to be satisfied together by heuristics [65].

It is worth mentioning that the connectivity property of the IIS plays an important role in IIS identification. An IIS is a connected subgraph in the context of constraint graph; if the construction of one IIS respects the connectivity among the constraints, the construction will be effective instead of being misled by adding irrelevant constraints or

constraints belonging to another IIS. When the first constraint of one IIS is identified, its adjacent constraints will be considered during IIS construction.

3.3.2 Verification phase

The construction phase ends when *MinConflict*() cannot find a partial solution on the critical constraints subset H. The constraint subset H will be handled by the verification phase for the unsatisfiability proof. *MAC*() sits within the verification phase, and attempts to find a solution exclusively for the constraints subset H. If such solution does not exist, we can conclude that the constraints in H form an IS.

Since the construction phase adopts an approximated approach, it is possible that the constraints in H may not sufficiently form an IS. In such case, *MAC*() will carry out a consistent assignment on H and the 2-phase algorithm will restart with the assignment as a partial solution for *MinConflict*() and enter again the construction phase.

3.3.3 Technique of saturation

In the construction phase, a specific procedure named *Saturate*() (Line 7) is adopted. It is an elegant technique which forms an induced subproblem with little computational cost.

Essentially, the saturation procedure automatically completes the constraints among the critical variables. The saturation technique travels all the critical variables and verifies its neighborhood variables to see if they are critical variables or not. If there is a neighborhood critical variable found, the saturation will covert the constraints exclusively on these critical variables into critical constraints.

The motivation is that there are a few Max-CSP solvers dealing with the minimization of variables violation instead of the constraints violation in the literature. By completing the constraints among the critical variables, any constraint-oriented Max-CSP solver can be loosely considered as a variable-oriented solver.

Another motivation is based on the fact that the constraints belonging to an IIS are identified one by one during the executions of *MinConflict*(). Without any acceleration technique, ideally the number of *MinConflict*() executions equals the number of constraints belonging to an IIS. With the saturation technique, the number of executions of *MinConflict*() can be reduced substantially.

The Example 3.7 illustrates the mechanism of saturation. The graph contains 4 nodes $\{A, B, C, D\}$ and 5 arcs $\{a, b, c, d, e\}$. As mentioned before, the *Selection*() will select only one constraint per iteration as critical constraint.

Suppose that after two iterations, the algorithm identifies two constraints a and b. With the saturation, the constraint d linking the two variables $\{A, C\}$ is converted to critical constraint automatically. Instead of three executions, the saturation accelerates such procedure by requiring one less run of *MinConflict*(). In more complicated cases, the number of executions can be reduced more quickly.

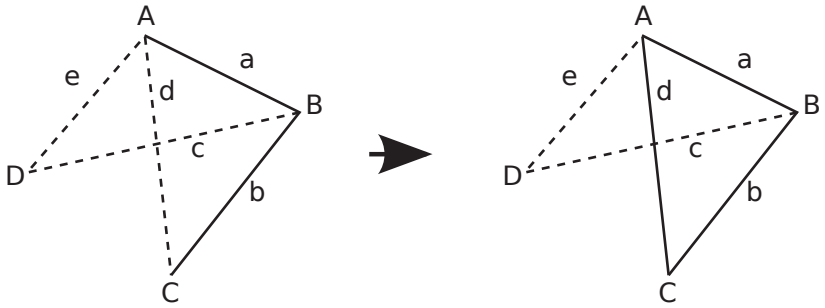

Fig. 3.7: Saturation

3.3.4 Preliminary analysis

The preliminary result of the above approach on the CELAR instances was reported in [96], accompanying with the results of the *wcore* proposed by Hemery et al. [61]. The experiments were carried out on a computer with 3Gb of memory and an Intel Core 2 Duo (T6300) 1.86GHz processor under Linux (Ubuntu 9.04) operating system. The benchmark is the well known telecommunication instances set CELAR. Several highest frequencies in benchmark are removed from the instances which makes the instances unsatisfiable. In Table 3.5, the first four columns describe the detail information of examined instances. The *HighestFreq* indicates the highest available frequency for each instance. The columns under *wcore* present the size of the IIS found by means of the number of variables and the number of constraints, and the computational time. The columns under *2-phase* indicate the minimal size of the IIS and minimal execution time in 50 runs obtained by *2-phase* algorithm, and the numbers in the parentheses are the maximal IIS size and maximal execution time in 50 runs.

It is noticed that in the first five instances, the numbers of constraints in IIS obtained by the *2-phase* are larger than the one found by *wcore*. The reason is that *2-phase* algorithm uses the saturation technique to complete all constraints among the variables. Overall, *2-phase* outperforms *wcore* by computational time on 3 instances for the same IIS of variables set size. Regarding the IIS size, *2-phase* found smaller IIS on 5 instances out of 10.

Scenario description				*wcore*			*2-phase*		
CELAR	Nb Vars	Nb Ctrs	Highest Freq	IIS vars	IIS ctrs	Time(s)	IIS vars	IIS ctrs	Time(s)
scen01	916	5548	666	10	25	828	10 (10)	44 (44)	102 (150)
scen02	200	1235	380	10	29	228	10 (10)	45 (47)	35 (45)
scen03	400	2760	380	9	28	208	9 (10)	35 (43)	59 (64)
scen04	680	3967	666	4	4	63	4 (5)	6 (11)	225 (292)
scen05	400	2598	666	4	4	40	5 (6)	6 (12)	162 (167)
scen06	200	1322	792	8	14	111	5 (5)	10 (12)	35 (55)
scen07	400	2865	792	9	16	112	5 (6)	10 (17)	98 (168)
scen08	916	5744	792	12	22	216	5 (8)	10 (25)	355 (480)
scen09	680	4103	792	9	21	150	5 (5)	10 (14)	210 (299)
scen11	680	4103	666	9	28	202	8 (10)	28 (57)	30 (70)

Table 3.5: Results comparison between *wcore* and *2-phase*

Based on the comparison, despite the 2-phase algorithm detecting the smaller IIS in variable sizes, its computational performance is not very impressive regarding Hemery et al.'s proposal. In this section, a computational consumption analysis is presented to analyze the algorithm design problem in the 2-phase algorithm.

Our first step is to generate a performance profile of the two phases of the *2-phase* algorithm which are construction and verification. *MinConflict*() is the main computation consumer in the construction phase, while *MAC*() plays the same role on the verification phase. The time consumption of both procedures is illustrated in Figure 3.8. It is quite obvious that the heuristic's runtime consumption largely surpasses the exact one. Through analysis, two main observations are revealed. Firstly, *MinConflict*() runs numerous times and always minimizes the fitness function on all constraints C. The design choice is made on the theoretical analysis on *Insertion* mentioned at the beginning of Section 3.3. Such a strategy avoids the risk of IIS destruction while it is a time consuming task. Further, comparing the number of violated constraints found by *MinConflict*() and the total number of constraints in one IIS, it suggests that there may exist more than one IIS inside the instance. The computation resource of the algorithm is used to search for one IIS among all possibilities in the full problem.

Secondly, in case of satisfiability found by *MAC*, the solution on H will be preserved and considered as an initial partial solution for the next run of the *2-phase* algorithm. There are two reasons behind the consistency on H. First, the constraints set H may intersect with or be included in an IIS. Second, the constraints set H may have no intersection with an IIS at all. In the first case, the constraints in H are not sufficient to form an IIS. While in the second case, the constraints in H have no interest in forming an IIS. By preserving the solution on H, the search efforts in both cases are taken into account.

These observations suggest that the construction phase need to be divided into two parts specifically to emphasize the location step: locating a constraint in one IS and constructing the IS around it. Instead of working on the entire

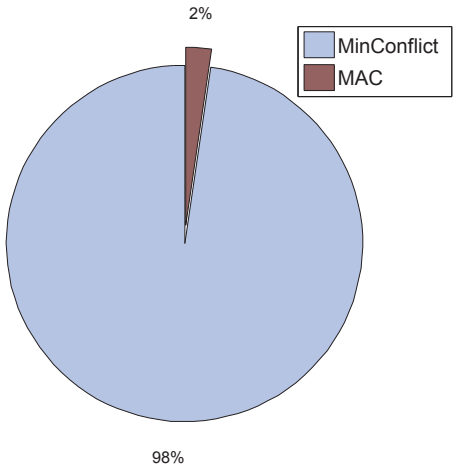

Fig. 3.8: Time consuming of first attempt on CELAR scen01

problem with costly procedure, the IS construction will focus only on a small subproblem. In the next section, a new general routine dedicated to IS identification on FAP instances will be introduced.

3.4 A general routine for Infeasible Subset (IS) identification - *LCV*

In this section, a general routine of IS identification will be introduced based on the drawbacks learned from previous experiments. The general routine consists of three independent components (see Figure 3.9) which are *Locator*, *Constructor* and *Verificator*.

These three components cooperate sequentially to identify an IS inside an over-constrained problem. The sequence can be described essentially as following: the *locator* scans the entire problem and attempts to locate a critical constraint potentially inside an IS; the *constructor* constructs a potential IS by adding one by one the adjacent constraints around the constraint identified by *locator*; and finally, the *verificator* provides the proof of infeasibility of the sub-

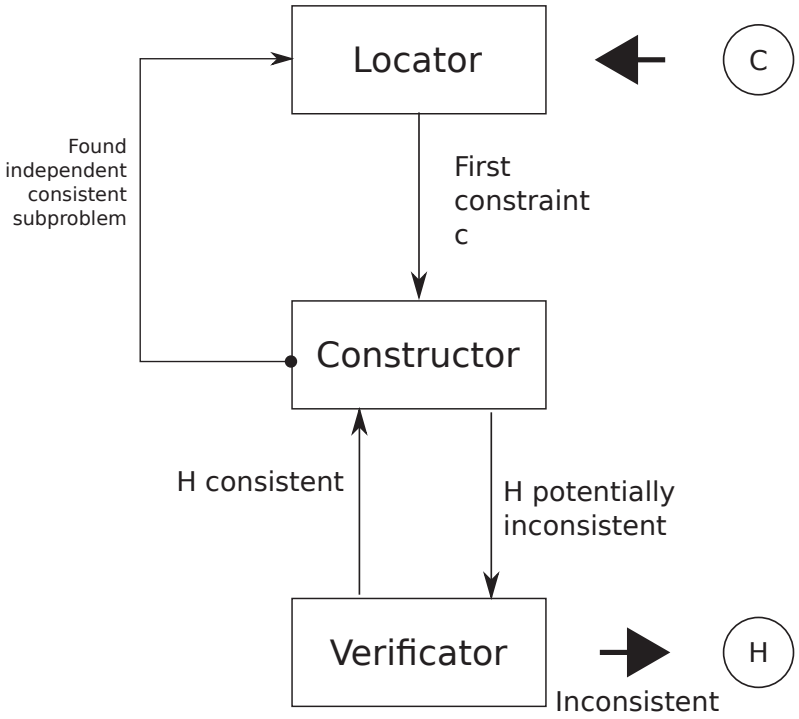

Fig. 3.9: *LCV* general routine

problem built by the *constructor*. In order to ease the description of this approach, we introduce the notion of a core which is a subproblem and potentially an IS.

It is noticed that when the unsatisfiability of core is proven, the core is an IS. The pseudo-code of *LCV* routine is illustrated in Algorithm 8, where X and C are the variables set and the constraints set of the problem, $C(x)$ and $X(c)$ indicate the constraints set on the variable x and the variables set under the constraint c, C_{Core} and X_{Core} denote the subset of constraints and the subset of variables forming a core, C_v is the violated constraint set, W_{C_v} is the weights on the violated constraints C_v generated by *BreakScan*() algorithm (described later).

The general *LCV* routine is mainly an IS identification algorithm, also, there are two subroutines dedicated to variables IIS and constraints IIS identification and they will be detailed in Section 3.5. In following sections, the three components in *LCV* will be separately explained.

Algorithm 8: *LCV*

input : a set of constraints C
output: an IS H

```
   /* ==================== Locator                                          */
1  if ArcConsistent(C) then
2  |  [W_{C_v}, C_v] ← BreakScan(C);
3  |  c ← Select(W_{C_v}, C_v);
4  else
5  |  x ← FindDeadend(X, C);
6  |  c ← SelectCtr(C(x));
7  end
   /* ==================== Constructor                                      */
8  X_Core ← X(c);
9  repeat
   |  /* Extension                                                         */
10 |  A_Core ← Assign(X_Core);
11 |  X_Core ← Extend(X_Core, A_Core);
12 |  if X_Core is feasible then return X_Core is a feasible problem;
13 |  H ← ∅,C_v ← ∅;
   |  /* Centralization                                                   */
   |  /* Extend() finds a deadend and fails to extend a partial solution  */
14 |  C_Core ← C(X_Core);
   |  /* The following loop is the same as construction phase in 2-phase
   |     algorithm                                                        */
15 |  repeat
16 |  |  if C_v ≠ ∅ then
17 |  |  |  c ← Select(C_v, H, W_{C_v});
18 |  |  |  H ← H ∪ {c};
19 |  |  |  H ← Saturate(H, C_Core);
20 |  |  end
21 |  |  [W_{C_v}, C_v, A_H] ← MinConflict(C_Core, H);
22 |  |  if C_v = ∅ then break;                    /* go to Line 31 */
23 |  until H ∩ C_v ≠ ∅ ;
24 |  X_Core ← X(H);
25 |  A_Core ← A_H;
26 until H ∩ C_v ≠ ∅ ;
   /* ==================== Verificator                                      */
27 A_H ← MAC(H);
28 if A_H infeasible then
29 |  return subset constraints H;
30 else
31 |  X_Core ← X(H);
32 |  A_Core ← A_H;
33 |  Goto Line 11;
34 end
```

3.4.1 Locator routine

At the end of Section 3.3.4, we have suggested that the IS search should be divided into two different procedures - IS location and IS construction. The motivation behind is that the IS identification should exclusively deal with the partial problem due to time consumption. In *LCV*, the *locator* sits right on the beginning of the routine to identify the location of an IS. It is described by Algorithm 9.

Algorithm 9: Locator

input : a set of constraints C
output: one constraint c
1 **if** $ArcConsistent(C)$ **then**
2 $\quad [W_{C_v}, C_v] \leftarrow BreakScan(C)$;
3 $\quad c \leftarrow Select(W_{C_v}, C_v)$;
4 **else**
5 $\quad x \leftarrow FindDeadend(X, C)$;
6 $\quad c \leftarrow SelectCtr(C(x))$;
7 **end**
8 **return** c

Based on above pseudo-code, the *locator* scans all constraints of the problem and identifies a specific constraint potentially inside an IS. As in the *2-phase* algorithm, such a constraint is selected among the violated constraints C_v according to the weights W_{C_v} (*Select*() in Line 3 of Algorithm 9). The weights W_{C_v} are carried out by *BreakScan*() which searches on entire problem. It is noticed that the *Select*() in the *2-phase* algorithm accepts three parameters C_v, W_{C_v}, H, while the *Select*() in *locator* only accepts two parameters C_v, W_{C_v}. These two functions will select the first constraint c exclusively on its weight in W_{C_v}. The difference is that after locating the first constraint c, *2-phase* starts to form H with the saturation process from c while *locator* stops immediately after finding c.

The slight difference is that after locating the first constraint, *2-phase* continues forming H by selecting another constraint c and *locator* stops to return directly the constraint c.

In the FAP instances, there may exist instances which are not consistent even at the Arc-Consistency level. Thus an Arc-Consistency algorithm *AC3* is deployed as a pre-filtering procedure. The integration of an AC algorithm is the significant difference between *locator* and the construction phase in the *2-phase algorithm*. *Locator* detects a *deadend* (see Definition 18) by *FindDeadend*() in case that the instance is not consistent at the Arc-Consistency level. In theory, a *deadend* variable is inside an IIS. The constraints which are adjacent with such a variable can be considered as a candidate list for the IIS location. Heuristically, the constraint c pruning the last value in the domain of *deadend*

variable will be chosen as the constraint potentially in an IIS by *SelectCtr*(). The *AC3* domain filtering is exclusively used in *locator*, and the domains of variables will be recovered after the location of the first constraint c.

For Arc-Consistent instances, the *AC3* pre-processing procedure reduces the variables domains which may accelerate search in *BreakScan*() [98]. From Figure 3.10, the acceleration on searching the IIS on FAP instances with AC pre-filtering is quite considerable.

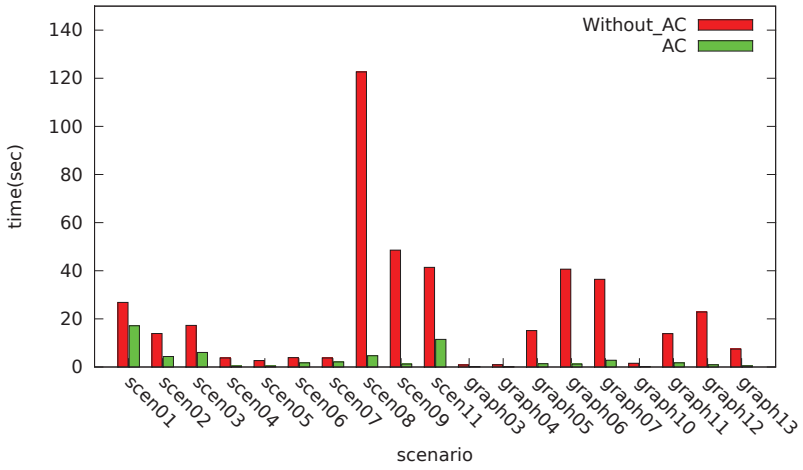

Fig. 3.10: Comparison of *LCV* IIS search speed with and without *AC3*

If the instance is Arc-Consistent, the *BreakScan* algorithm will sit right after the *AC3* algorithm to locate the first constraint. Since in *LCV* only one constraint need to be located, it essentially requests the algorithm to scan all the constraints and proposes a specific constraint which addresses the location of a potential IIS. Considering it is the only component working on the entire problem, the *locator* algorithm should not be costly on time computing.

Algorithm 10 slightly changes the original approach of *prefiltering* by embedding a weighting system. The weights W_{C_v} on the violated constraints in C_v are generated by *increaseOneUnitWeight*() procedure during each iteration of *BreakScan* with initialization to zero. The selected critical constraint c is the constraint with the maximal weight in W_{C_v}. Using a weighting system as a statistic learning technique is not a new idea in CSP resolution, despite its simplicity, it intentionally guides the search toward the subset of critical constraints which are difficult to satisfy together. The procedure stops when the *localSearch*() cannot find a solution satisfying all critical constraints in H.

The fitness function of *localSearch*() can be expressed as:

$$f_{localSearch} = \sum_{i \in C} \psi_i \times W_i \tag{3.9}$$

Algorithm 10: BreakScan

input : a set of constraints C
output: a set of constraints C_v and a set of weights W_{C_v}

```
1  W_{C_v}, H, C_v ← ∅;
2  while H ∩ C_v = ∅ do
3      if C_v ≠ ∅ then
4          W_{C_v} ← inceaseOneUnitWeight(C_v, W_{C_v});
5          H ← H ∪ C_v;
6      end
7      C_v ← localSearch(C, W_{C_v}, H);
8  end
9  C_v ← violatedCtrs(H);
10 return C_v, W_{C_v}
```

where ψ_i is a boolean variable on the constraint c_i, it equals 1 if the constraint c_i is violated, 0 otherwise; W_i is the weight of the constraint c_i. By respecting the function, *localSearch*() attempts to minimize the violation on highly weighted constraints. When *localSearch*() stops, it returns a set of violated constraints C_v which is equal to the final set H. *localSearch*() stops after a given number of iteration fixed to 1000 without improvement. The local search chosen to be embedded in *BreakScan* needs to be efficient.

As mentioned before, *TabuCol* can be considered as candidate for such role. It is interesting to compare it with an even simpler local search and to see the impact between the quality of local search and the performance of *BreakScan*(). The simple local search we used for such comparison is a hill climbing local search which is created by simply removing the tabu list in *TabuCol*. The Figure 3.11 shows the results reported in [99], which demonstrates the performance gains by replacing *TabuCol* by hill climbing. All the time consuming comparison is based on running entire *LCV* algorithm with *TabuCol* or hill climbing heuristic as *localSearch*() in *BreakScan*() with same size of IS found.

The experiment is conducted without the *AC3* pre-filtering procedure. It shows that the simple local search approach has performance gains on 15 instances out of 19. We can conclude that even a simple Hill Climbing local search can guarantee performance of *LCV* to search an IS under the guidance of the *prefiltering* algorithm. The future experiments will be carried out exclusively with the Hill Climbing as *localSearch*() in *locator*.

3.4.2 Constructor routine

When the critical constraint c is identified by *locator*, the *constructor* is in charge of building a core around it. From the experiment in [96], the *2-phase* algorithm gave no impressive results in comparison with *wcore* approach. It may be due to that *wcore* adopts a constructive approach in IIS construction while the *2-phase* algorithm always works

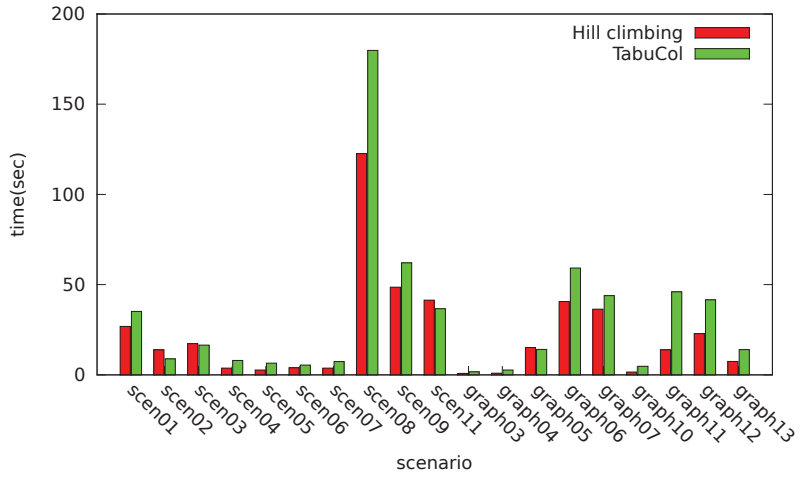

Fig. 3.11: Comparison of *LCV* processing time with hill climbing and *TabuCol* as *localSearch*() in *BreakScan*()

on the entire problem. By dividing IS construction into *locator* and *constructor*, we may improve the performance. Algorithm 11 is the routine of *constructor* which is a combination between an extension procedure and a centralization procedure.

Firstly, we randomly assign the two variables of the initial constraint c with two consistent values to initialize $\mathcal{A}_{Core}$. The extension *Extend*() works like *DSatur*, which attempts to assign one by one the variables adjacent with the core X_{Core}. The variable selection is based on MRV (Minimum Remaining Values) instead of *DSatur*'s color saturation degree, and one available value in its domain will be assigned to the selected variable which is consistent with the partial solution $\mathcal{A}_{Core}$ on X_{Core}. The procedure stops when *Extend*() finds a *deadend* variable, the *deadend* variable will be added in X_{Core}. *Extend*() returns the variables set X_{Core} and the partial solution of the core. In case *Extend*() finds a feasible core, *constructor* will return the core as a subproblem (see Line 5).

The centralization procedure is executed right after the extension to extract H from X_{Core} issued from extension. The centralization procedure uses the *prefiltering* algorithm in [78] to extract a critical constraint subset H from the core constraints C_{Core}. It stops when it cannot find a solution $\mathcal{A}_H$ on H. i.e. there is at least one constraint violated in H. In case that there is a feasible solution on C_{Core} found during the centralization procedure, the *constructor* will continue to extend the core by *Extend*().

Our preliminary analysis shows that the maximal constraint and variable size reduction reaches 80% on CELAR and GRAPH instances with the centralization procedure. Comparing the bold lines in Figure 3.12, the first represents a core C_{Core} after the extension procedure and the latter is a critical constraint subset H processed by the centralization procedure. We see that with the presence of the centralization procedure, the size of the core is greatly reduced to an

Algorithm 11: Constructor

input : a set of constraints C, a located constraint c and a set of weights W_{C_v}
output: the core constraint set H

1 $X_{Core} \leftarrow X(c)$;
2 **repeat**
/* Extension */
3 $\mathcal{A}_{Core} \leftarrow Assign(X_{Core})$;
4 $X_{Core} \leftarrow Extend(X_{Core}, \mathcal{A}_{Core})$;
5 **if** X_{Core} *is feasible* **then return** X_{Core} *is a feasible problem;*
6 $H \leftarrow \emptyset$,$C_v \leftarrow \emptyset$;
/* Centralization */
/* $Extend()$ finds a *deadend* and fails to extend a partial solution */
7 $C_{Core} \leftarrow C(X_{Core})$;
/* The following loop is the same as construction phase in *2-phase* algorithm */
8 **repeat**
9 **if** $C_v \neq \emptyset$ **then**
10 $c \leftarrow Select(C_v, H, W_{C_v})$;
11 $H \leftarrow H \cup \{c\}$;
12 $H \leftarrow Saturate(H, C_{Core})$;
13 **end**
14 $[W_{C_v}, C_v, \mathcal{A}_H] \leftarrow MinConflict(C_{Core}, H)$;
15 **if** $C_v = \emptyset$ **then** break; /* go to Line 17 */
16 **until** $H \cap C_v \neq \emptyset$;
17 $X_{Core} \leftarrow X(H)$;
18 $\mathcal{A}_{Core} \leftarrow \mathcal{A}_H$;
19 **until** $H \cap C_v \neq \emptyset$;
20 **return** H;

acceptable scope for the unsatisfiability check. Comparing to the Algorithm 7, the new cooperation between *locator* with *constructor* has significant performance gains even without the Arc-Consistency check in *locator*. Figure 3.13 shows the time consumption between the *2-phase* algorithm and the *LCV* algorithm. The performance improvement in time is significant on the CELAR instances, *LCV* outperforms the *2-phase* algorithm on all 10 instances with the same verification algorithm inside them. For the performance on IIS of variables size, *LCV* only found a larger IIS on 3 instances out of 10 in CELAR (see Table 3.5 and Table 3.6).

3.4.3 Verificator routine

The *verificator* is the final component in *LCV* which provides the infeasibility/feasibility proof of the critical constraints subset H. The verificator is implemented by an exact search algorithm.

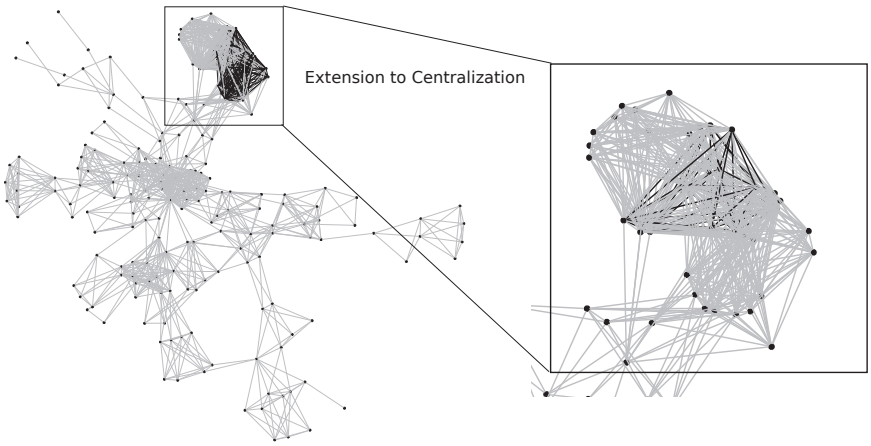

Fig. 3.12: CELAR scen02 extension to centralization

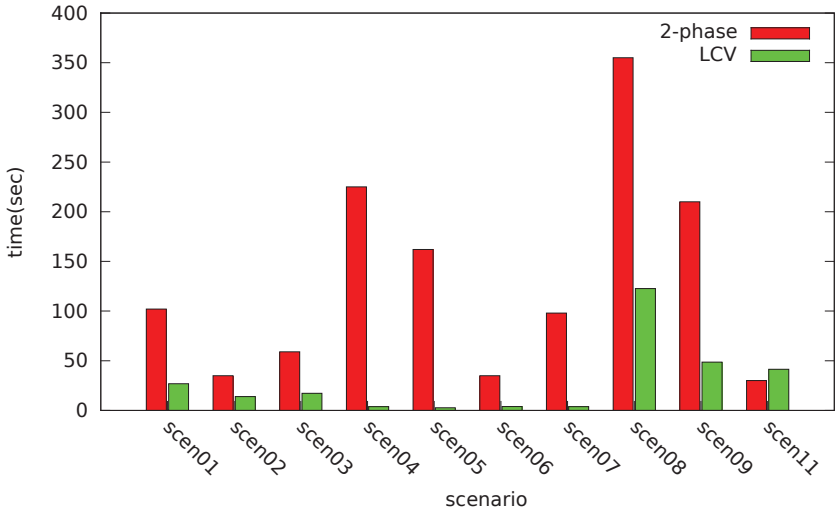

Fig. 3.13: Time consuming between *2-phase* and *LCV*

In Section 1.4, several exact algorithms were described and compared briefly. Among them, the *MAC* algorithm is a good candidate to act as a verificator thanks to its effectiveness [30] and simplicity in implementation. It consists of embedding an Arc-Consistency propagator inside a chronological backtracking routine. There is no overhead memory management thanks to the effectiveness and efficiency of the propagator. As shown in Figure 3.9, the arrow from *verificator* to *constructor* indicates the correctness routine if the feasibility on H is proven and the critical constraints subset H will be returned to *constructor*. A new core will be constructed around X_{Core}. This routine is totally different from the restart mechanism adopted in Algorithm 7, it guarantees that the previous effort will be used. Thanks to

centralization in *constructor*, even if the core X_{Core} grows, its size can still be reduced by the centralization process. If *verificator* proves the unsatisfiability of H, *LCV* returns H as IS.

3.5 IIS of variables and IIS of constraints

In previous sections, we have presented *LCV* as an IS identification algorithm. Based on the result of *LCV*, we now intend to identify an IIS inside the extracted IS. Following the definition of IIS, it is noticed that an IIS can be identified as an IIS of variables or an IIS of constraints (see Definition 22 and Definition 23).

Based on IIS definition (see Definition 21), any IIS can be identified by removing one by one either the variable or the constraint from an IS. We attempt to iteratively identify a smaller IS inside one IS until no smaller IS can be found. The routine of IIS of variables and IIS of constraints detectors can be briefly described in Figure 3.14. The two detectors iteratively consider the output IS of the previous stage as input IS for the current stage. The procedure stops when the size of variables or constraints set cannot be reduced. In the next section, two detectors named IIS of variables and IIS of constraints detector will be detailed.

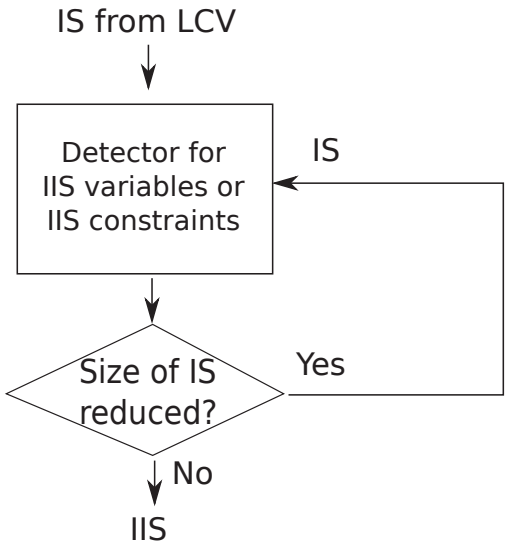

Fig. 3.14: Routine of IIS detectors

3.5.1 IIS of variables detector

Based on the pseudo-code in Algorithm 12, the detector can be divided into two parts, the locate core and construct core. The IIS of variables detector firstly locates a small critical constraint subset H inside the entering IS. The locate core procedure is the implementation of *prefiltering* in [78] with the embedded saturation technique. The saturation procedure (see *Saturate*()) is used to complete all the constraints among the critical variables.

Secondly, the detector iteratively adds one x into the located core formed by the critical variables X_H and the critical constraints H. The variable selection is based on the MRV heuristic. The procedure stops when *MAC*() algorithm proves that the core is unsatisfiable.

Since the algorithm uses heuristics to add the variables into the core, the final output can be only considered as an approximated IIS. The detector will be executed iteratively until the variables set sizes of input IS and output approximated IIS are equal.

Algorithm 12: IIS of variables detector

input : an IS $Core$
output: an approximated IIS of variables X_H

$H \leftarrow \emptyset$,$C_v \leftarrow \emptyset$;
/* locate core */
repeat
  $H \leftarrow H \cup C_v$;
  $H \leftarrow Saturate(H, Core)$;
  $C_v \leftarrow MinConflict(Core, H)$;
until $C_v \cap H \neq \emptyset$;
$X_H \leftarrow X(H)$;
/* construct core */
while $MAC(H)$ *feasible* **do**
  if *there is no* $c \in X(Core) \setminus X_H$ **then return** X_H;
  $x \leftarrow MRV(X(Core) \setminus X_H)$;
  $X_H \leftarrow X_H \cup \{x\}$;
end
return X_H;

3.5.2 IIS of constraints detector

Following the strategy adopted in [78], the IIS of constraints will be identified on the basis of IIS of variables. When IIS of variables identifies an approximated IIS, IIS of constraints detector will locate a core exclusively based on the

critical constraints H without saturation. After the core is located, the detector attempts to iteratively add one constraint into the core until the core H becomes infeasible.

Regarding the pseudo-code in the locate core procedure of Algorithm 13, the slight difference from IIS of variables detector is the absence of saturation. In construct core procedure, the constraint c is chosen among the constraints set adjacent with the variable chosen by MRV (see *MAC*()). In order to minimize the size of the constraints set, the saturation technique is not applied here.

Algorithm 13: IIS of constraints detector

input : an IS $Core$
output: an approximated IIS of constraints H

1 $H \leftarrow \emptyset$,$C_v \leftarrow \emptyset$;
/* locate core */
2 **repeat**
3 | $H \leftarrow H \cup C_v$;
4 | $C_v \leftarrow MinConflict(Core, H)$;
5 **until** $C_v \cap H \neq \emptyset$;
/* construct core */
6 **while** $MAC(H)$ *feasible* **do**
7 | **if** *there is no* $c \in Core \setminus H$ **then return** H;
8 | $c \leftarrow MRV(Core \setminus H)$;
9 | $H \leftarrow H \cup \{c\}$;
10 **end**
11 **return** H;

Both detectors can still provide high performance thanks to the size of critical constraints subset H, it is noticed that *MAC* is heavily executed. In the next section, the experimental results conducted on IIS identification by *LCV* and the two IIS detectors will be detailed and analyzed.

3.6 Experimental results

For both *LCV*, and *wcore* [61], proposed by Hemery et al., experimental results were carried out on an Intel Core 2 Due E5300 (2.6GHz) machine with 3.2Gb memory under linux (Ubuntu distribution). The proposed approach is implemented in C++ and compiled by GCC. The reference method *wcore* proposed by Hemery et al. is coded in Java. The minimum and average execution times on each instance are measured for 5 executions. All results presented as IIS are an approximated IIS.

3.6.1 Results for CELAR and GRAPH

All the results here are carried out on the CELAR and GRAPH benchmarks by *LCV* and *wcore* algorithm. The version of *wcore* is the faster version found on the authors' website which is different from the version used in Section 3.3.4. Table 3.6 gives the results on IS and IIS of variables obtained by *LCV*. Table 3.7 compares the results of IIS of constraints obtained by both algorithms. In Table 3.7, the computational time obtained by *wcore* is measured by one run on each instance since it is a deterministic algorithm. For both Table 3.6 and Table 3.7, the computational times obtained by *LCV* are indicated separately by minimum runtime and average runtime obtained in 5 runs for each instance. In Table 3.7, *wcore* fails to find an IIS on graph03 and graph10 (represented by "–"). The success rate of *LCV* on CELAR/GRAPH is 100% for all instances.

	LCV IS				*LCV*+detector IIS Var			
CELAR	Var	Ctr	Min(sec)	Ave(sec)	Var	Ctr	Min(sec)	Ave(sec)
scen01	10(12.4)	44(56.4)	17.16	18.77	10(10)	44(44)	17.9	20.62
scen02	10(12)	45(64.4)	4.38	5.39	10(11)	45(54.4)	7.01	7.61
scen03	10(10)	43(44.2)	6.05	6.08	10(10)	43(44.2)	6.82	7.72
scen04	4(4)	6(64.4)	0.36	0.36	4(4)	6(6)	0.55	0.59
scen05	4(4)	6(64.4)	0.36	0.36	4(4)	6(6)	0.59	0.63
scen06	5(5)	10(10)	1.73	1.94	5(5)	10(10)	2.04	2.41
scen07	8(8.4)	22(24.8)	2.12	3.99	8(8)	22(22)	3.19	5.81
scen08	9(12)	26(42.7)	4.69	5.33	7(7.7)	17(23)	7.39	9.24
scen09	7(9.4)	19(33.8)	1.22	3.02	5(6.5)	10(17)	3.85	4.81
scen11	8(8)	28(28)	11.48	12.23	8(8)	28(28)	12.43	13.54
	LCV IS				*LCV*+detector IIS Var			
GRAPH	Var	Ctr	Min(sec)	Ave(sec)	Var	Ctr	Min(sec)	Ave(sec)
graph03	4(5.2)	4(7.4)	0.19	0.43	4(4)	4(44)	0.54	0.81
graph04	3(3)	2(24.8)	0.15	0.17	2(2)	1(10)	0.15	0.17
graph05	7(7.8)	9(12.2)	1.36	1.36	7(7.7)	9(9)	1.36	1.36
graph06	7(7.4)	17(23.2)	4.25	4.34	4(6.6)	6(8)	4.46	8.27
graph07	12(19)	28(29.2)	2.81	4.03	4(4.5)	6(8)	3.97	4.43
graph10	2(2)	1(10)	0.2	0.2	2(2)	1(10)	0.19	0.19
graph11	6(9.4)	15(23.6)	1.73	2.63	4(5.2)	6(11.2)	3	3.22
graph12	6(6.8)	11(11.4)	0.93	3.19	5(5)	9(9)	3.52	4.91
graph13	4(4)	6(64.4)	0.46	0.54	4(4)	6(6)	0.69	0.81

Table 3.6: Results on IS and IIS of variables obtained by *LCV*

In Table 3.6, the Var and Ctr columns represent the minimal IS and IIS sizes for all instances on 5 runs, and the average size of IS and IIS is in parentheses. For most of the instances in GRAPH, the *LCV*+IIS of variables detector algorithm highlights its effectiveness on variables and constraints reduction.

It is noticed that *LCV* alone is quite effective on IIS variables identification. *LCV*+detector only finds 7 instances with smaller size of IIS than the IS found by *LCV* (see highlighted numbers under column sixth and seventh). Among these 7 instances, only graph06 and graph07 have significant improvement on IIS size. The time consumption for the IIS of variables detector is minor when we compare the average time of *LCV* alone and the average time of the *LCV*+detector, the maximal time consumption of the detector occurs on instance scen01 with less than 2 seconds for the same size between IS and IIS.

Table 3.7 shows the IIS of constraints size and computational time comparison between *LCV*+detector and *wcore*. The minimal size of IIS of constraints found on 5 runs is shown under the columns Var and Ctr and the average size found on 5 run is shown in parentheses. Regarding the computational time, *LCV* outperforms *wcore* on all instances except 2 (graph06 and graph07) on average time in 5 runs, its minimum time surpasses *wcore* on all instances. Regarding the number of constraints in the IIS of constraints, *wcore* obtains a smaller number of constraints on only 4 instances out of 19. Among them, two instances (graph06 and graph07) have larger variables numbers in the IIS of constraints than the one found by *LCV*.

Table 3.8 extracts the results of the IIS of variables and the IIS of constraints obtained by *LCV*+detector from Table 3.6 and Table 3.7. It is noticed that the IIS of constraints detector can still effect the results of IIS of variables. For all 19 instances, the IIS of constraints detector reduces the number of constraints on 10 instances. On the other hand, with the IIS of variables detector alone, we can always get an IIS with relatively small size.

Figure 3.15 shows *LCV*'s average runtime is very good comparing to the *wcore* approach except in two instances. Referring to the minimal runtime in Table 3.7, the *LCV*'s performance surpasses the compared method on all instances and the maximal gain reaches 95%.

In Figure 3.16, we plot the performance gains on the sizes of the IIS for the number of variables in red and the number of constraints in green between *LCV* and *wcore*. The performance loss can only be found on 4 instances out of 19, particularly when we compare the constraints sizes. The sizes of variables in IIS obtained by *LCV* are either equal or less than the one found by *wcore*.

Next we generate *LCV* algorithm profile of runtime on two CELAR instances. Figure 3.17 shows the runtime profile of three components in *LCV* on instance scen01, Figure 3.18 shows the runtime profile on instance scen08. It needs to notice that the scen01 is consistent at the Arc-Consistency level, while scen08 is inconsistent. Comparing Figures 3.17 and 3.18, it is obvious that *locator* consumes the major part of runtime on scen01, while on scen08, the *constructor* is more involved. Since scen08 is not consistent at the AC level, the AC3 algorithm in *locator* can efficiently find a *deadend* variable and locate an IIS, while on scen01, *locator* needs to call *Breakscan* to locate the IIS which is a more time consuming procedure. In scen08, thanks to the location of the *deadend* variable, the unsatisfiability of IIS

CELAR	*wcore* IIS Ctr Var	Ctr	Time(sec)	*LCV*+detector IIS Ctr Var	Ctr	Min(sec)	Ave(sec)
scen01	10	25	30.33	10(10)	27(27.2)	19.47	22.49
scen02	10	29	18.02	10(11)	18(23)	9.42	10.98
scen03	10	29	20.47	10(10)	18(19.2)	10.21	12.22
scen04	4	4	9.87	4(4)	4(5)	1.07	1.95
scen05	4	4	7.81	4(4)	5(5.2)	1.22	1.31
scen06	8	14	17.91	5(5)	7(7.2)	2.65	3.25
scen07	9	16	17.22	8(8)	16(17.3)	7.05	7.95
scen08	12	22	22.07	7(7.7)	13(17)	8.76	11.5
scen09	7	14	14.97	5(6.5)	10(14.7)	4.18	6.2
scen11	8	28	22.88	8(8)	28(28)	13.95	15.12
GRAPH	*wcore* IIS Ctr Var	Ctr	Time(sec)	*LCV*+detector IIS Ctr Var	Ctr	Min(sec)	Ave(sec)
graph03	–	–	–	4(4)	3(3.4)	0.85	0.95
graph04	4	3	5.27	2(2)	1(1)	0.19	0.21
graph05	8	11	10.96	7(7.8)	9(12.2)	1.36	1.36
graph06	5	5	9.6	4(4.3)	6(7.2)	5.8	10.33
graph07	5	5	9.53	4(4.3)	6(7)	5.15	10.25
graph10	–	–	–	2(2)	1(1)	0.22	0.23
graph11	6	6	11.43	4(5.2)	6(7.4)	3.24	3.83
graph12	7	8	12.04	5(5.2)	7(7.6)	4.25	5.79
graph13	10	17	20.79	4(4)	6(6)	0.94	1.05

Table 3.7: Results comparison between *wcore* and *LCV* on IIS of constraints

is not very difficult to prove. In scen01, *verificator* uses significant time to prove the unsatisfiability on a smaller over constrained critical constraints subset.

From the comparison, the proposed *LCV*+detector algorithm surpasses the performance of *wcore* on both IIS size and computational time. On the instances consistent at the AC level, as mentioned before, *wcore* consists of embedding a variable selection heuristic inside a MAC (Maintaining Arc-Consistency during search). It suffers from the ordering of assigned variables in its MAC backtracking algorithm. *LCV* adopts a totally different approach which scans all constraints of the entire problem and proposes a critical constraint potentially inside an IIS which avoids the impact of variables ordering during the search. For the instance inconsistent at AC level, *LCV*'s *locator* can efficiently locate a *deadend* variable which is absolutely inside an IIS. Such a strategy dramatically improves the computational performance.

CELAR	IIS Vars Var	Ctr	IIS Ctr Var	Ctr	GRAPH	IIS Vars Var	Ctr	IIS Ctr Var	Ctr
scen01	10	44	10	27	graph03	4	4	4	3
scen02	10	45	10	18	graph04	2	1	2	1
scen03	10	43	10	18	graph05	7	9	7	9
scen04	4	6	4	4	graph06	4	6	4	6
scen05	4	6	4	5	graph07	4	6	4	6
scen06	5	10	5	7	graph10	2	1	2	1
scen07	8	22	8	16	graph11	4	6	4	6
scen08	7	17	7	13	graph12	5	9	5	7
scen09	5	10	5	10	graph13	4	6	4	6
scen11	8	28	8	28					

Table 3.8: IIS of variables and IIS of constraints detectors comparison

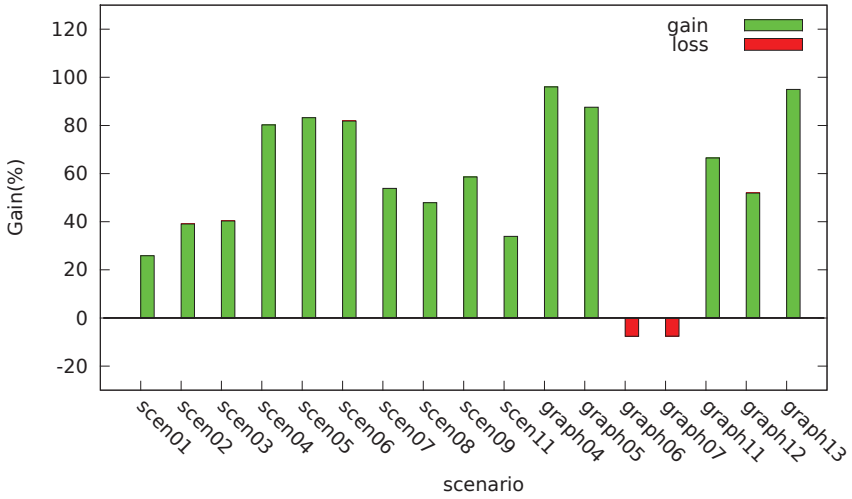

Fig. 3.15: *LCV*+IIS of constraints detector on average runtime compared to *wcore*

3.6.2 Results for ROADEF2001 challenge

In this section, we give the results on the instances of ROADEF2001 challenge which were carried out by our *LCV* with both IIS variables and IIS of constraints detectors. We also used *wcore* on these instances but it failed to find any IIS in 1000 seconds. Thus we will only present the computational results obtained by our approach.

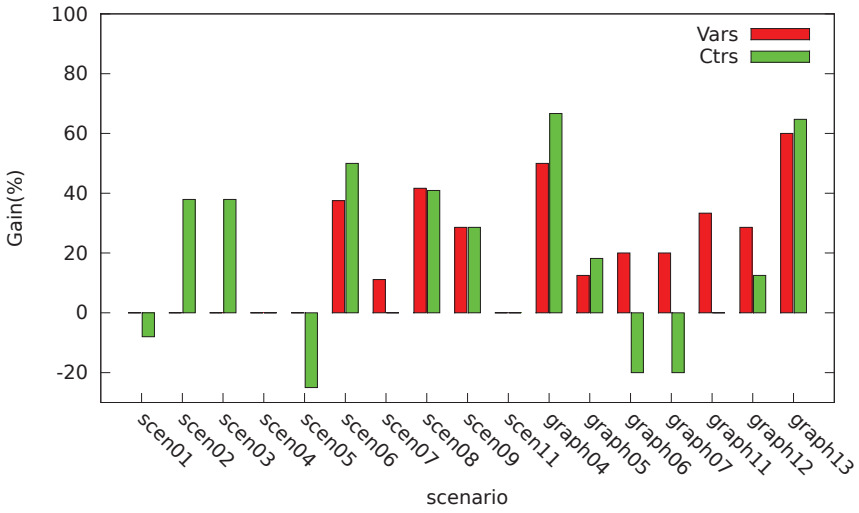

Fig. 3.16: *LCV* IIS of constraints size gain compared to *wcore*

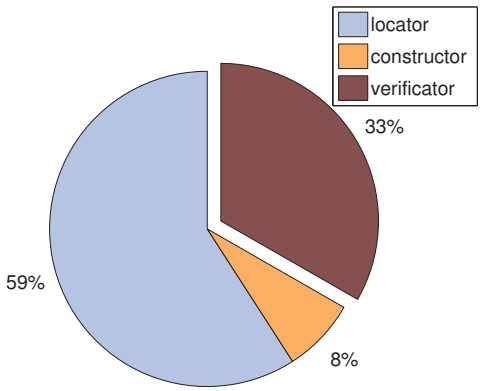

Fig. 3.17: CELAR scen01 *LCV* profile

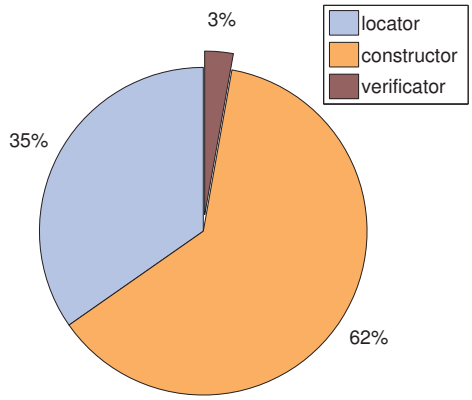

Fig. 3.18: CELAR scen08 *LCV* profile

Regarding all the results based on minimum IIS found (see Figure 3.19), *LCV* fails to find an IIS on 4 instances out of a total of 220 instances (N/A results represented by ”–”). The success rate of *LCV* on 216 instances is 100% on 5 runs. On 77 of the 216 success instances, the IIS of variables detector extracts an IIS size less than the size of IS found by *LCV* (orange part in left histogram). The right histogram presents the results of IIS constraints detector on the basis of IIS of variables detector results on these 77 instances. For 18 of those 77 instances, IIS of constraints detector reduces the constraints set size on the basis of IIS of variables found by IIS of variables detector (represented by IIS constraints reduction). It is noticed also with IIS of constraints detector that there are 21 of those 77 instances which even extract smaller IIS of variables set size from IIS of variables detector results. All improvements from the

IIS constraints detector are highlighted in Tables 3.10 and 3.11. These 21 instances indicate that the IIS of variables detector loses its effectiveness on IIS detection. Sometimes, the IIS of variables detector did not find an IIS, instead it finds an IS. The strategy of adding variables to core in the IIS of variables detector is not always effective when compared to the removing variables technique described in the literature [62].

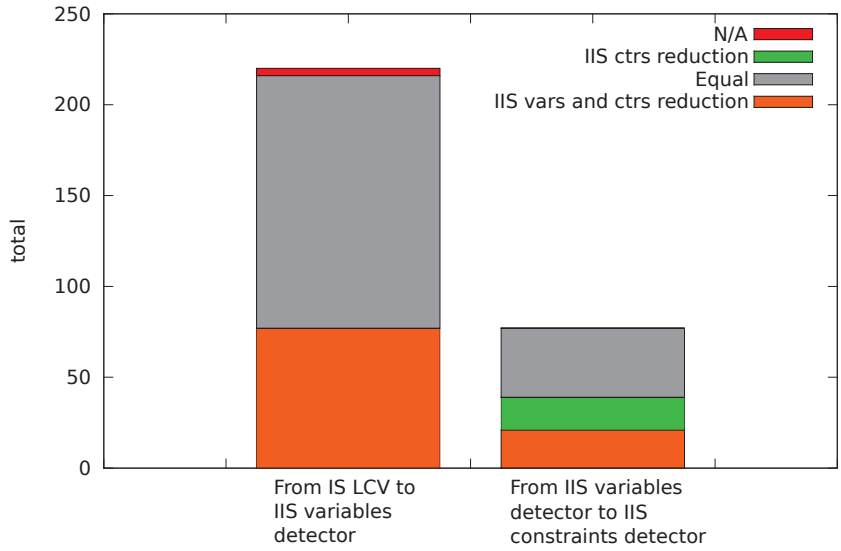

Fig. 3.19: Overall performance on IIS size for ROADEF2001

Regarding all results on the 40 scenarios including different levels (from fapp01 to fapp40), they can be classified into two categories (see Table 3.9). There are 16 scenarios in first class A, whose sizes of IIS are increasing with the increasing restriction level despite a few exceptions. For the class B of 24 scenarios, the sizes of IIS are either decreasing with the increasing restriction level, or not related to the level. Since restriction level 0 is the most strict one and the variables are strongly under constrained, it explains that the situation happened in class A. The IIS grows its size when the constraint relaxation is taking place. Class B contains the scenarios whose size of IIS decreases with the increasing of level. This is mainly caused by *locator*. With the relaxation of restriction level, the search locates different IIS inside these problems. So we can identify that there are different categories of problem property in ROADEF2001 instances for IIS identification.

Tables 3.10 and 3.11 show all results obtained by *LCV* on the ROADEF2001 instances at their infeasible levels. Under the column ”instance level”, the name of the instances and the levels examined are listed. The three main columns - IS, $IISvars$ and $IISctrs$ represent the results obtained by *LCV* alone, *LCV* with the IIS of variables

Scenario	Class	Scenario	Class	Scenario	Class	Scenario	Class
fapp01	A	fapp11	A	fapp21	B	fapp31	B
fapp02	B	fapp12	B	fapp22	B	fapp32	A
fapp03	B	fapp13	B	fapp23	B	fapp33	A
fapp04	B	fapp14	B	fapp24	B	fapp34	B
fapp05	A	fapp15	B	fapp25	A	fapp35	A
fapp06	B	fapp16	A	fapp26	B	fapp36	B
fapp07	B	fapp17	B	fapp27	A	fapp37	B
fapp08	A	fapp18	B	fapp28	B	fapp38	A
fapp09	A	fapp19	A	fapp29	B	fapp39	A
fapp10	A	fapp20	A	fapp30	B	fapp40	B

Table 3.9: Results classification of ROADEF2001

detector and *LCV* with IIS of constraints detector on the basis of IIS of variables. The column V indicates the number of variables, C indicates the number of constraints and T the average computing runtime in seconds on 5 runs.

instance	level	IS V	IS C	IS T(sec)	IIS vars V	IIS vars C	IIS vars T(sec)	IIS ctrs V	IIS ctrs C	IIS ctrs T(sec)
fapp01	0	8	12	6.26	3	3	6.82	3	3	7.03
	1	7	9	10.71	7	9	12.92	7	9	15.66
	2	14	19	110.43	7	7	113.12	7	7	114.04
	3	16	19	75.29	7	7	87.71	7	7	88.6
fapp02	0	3	3	1.6	3	3	1.83	3	3	1.98
	1	3	3	2.94	3	3	3.08	3	3	3.31
fapp03	0	8	8	4.29	8	8	6.21	8	7	7.16
	1	7	8	3.42	7	8	3.88	6	5	5.03
	2	4	4	3.24	3	3	4.18	3	3	4.65
	3	3	3	2.66	3	3	2.81	3	3	2.97
	4	3	3	3.99	3	3	4.22	3	3	4.44
	5	3	3	4.98	3	3	5.21	3	3	5.37
	6	3	3	9.8	3	3	10.03	3	3	10.18
fapp04	0	3	3	5.05	3	3	5.46	3	3	5.89
fapp05	0	9	9	8.17	9	9	17.52	6	5	19.37
	1	11	12	21.56	11	12	29.77	11	11	33.29
	2	17	18	27.63	7	7	62.75	7	7	63.93
	3	13	15	13.43	11	13	36.15	11	11	58.36
	4	15	24	25.84	7	7	30.46	7	7	31.33
	5	16	20	33.62	7	7	52.5	7	7	53.77
	6	9	9	43.55	9	9	65.26	9	9	86.88
	7	14	16	86.55	9	10	90.05	7	7	93.53
	8	14	16	94.31	14	16	101.87	14	14	120.02
	9	19	23	72.56	12	12	86.88	12	12	91.1
	10	16	20	57.05	15	18	74.78	12	12	96.69
fapp06	0	4	4	4.77	3	3	5.28	3	3	5.44
	1	3	3	8.87	3	3	9.03	3	3	9.27
	2	4	4	10.89	3	3	11.45	3	3	11.68
	3	3	3	15.31	3	3	15.46	3	3	15.7
	4	10	10	73.75	3	3	75.9	3	3	76.52
fapp07	0	3	3	1.85	3	3	2.14	3	3	2.43
	1	4	3	4.01	4	3	5.66	4	3	6.16
	2	5	4	5.17	5	4	41.25	5	4	41.78
	3	4	3	2.73	4	3	49.11	4	3	49.62
	4	4	3	7.02	4	3	10.17	4	3	10.56
	5	6	5	4.51	5	4	6.7	5	4	7.47
	6	3	3	6.51	3	3	6.74	3	3	6.97
	7	4	5	3.47	3	3	3.91	3	3	4.06
	8	3	3	6.61	3	3	7.7	3	3	8.52
fapp08	0	3	3	6.84	3	3	7.24	3	3	7.4
	1	3	3	5.88	3	3	6.03	3	3	6.18
	2	3	3	5.68	3	3	5.83	3	3	5.98
	3	6	6	21.5	5	5	26.71	5	5	29.38
	4	7	7	52.41	7	7	52.99	7	7	53.47
fapp09	0	5	4	18.29	3	3	19.16	3	3	19.46
	1	8	7	68.57	3	3	69.7	3	3	70.11
	2	7	7	21.13	4	4	22.44	4	4	22.77
fapp10	0	5	4	11.91	5	4	13.27	5	4	13.97
	1	5	5	13.29	5	5	13.72	5	5	14.33
	2	3	3	17.53	3	3	17.69	3	3	17.94
	3	4	4	8.29	4	4	10.01	4	4	10.36
	4	5	5	37.71	5	5	37.94	5	5	38.25
	5	8	8	32.13	6	6	32.89	6	6	33.33
fapp11	0	4	3	2.64	4	3	4.2	4	3	5.39
	1	3	3	11.41	3	3	13.42	3	3	14.19
	2	3	3	20.34	3	3	21.11	3	3	21.66
	3	3	3	16.63	3	3	17.18	3	3	17.74
	4	7	6	57.08	5	5	60.18	5	5	60.88
	5	3	3	34.08	3	3	34.62	3	3	35.18
fapp11	6	3	3	30.85	3	3	31.87	3	3	32.42
	7	6	6	51.82	6	6	52.46	6	6	53.1
fapp12	0	3	3	13.2	3	3	13.87	3	3	14.8
	1	3	3	96.61	3	3	97.34	3	3	98.08
fapp13	0	9	10	7.93	3	3	8.77	3	3	8.97
	1	3	3	10.72	3	3	11.04	3	3	11.35
	2	3	3	9.73	3	3	10.05	3	3	10.27
fapp14	0	5	5	53.72	4	4	54.35	4	4	54.6
	1	13	13	163.46	7	7	164.69	7	7	165.05
	2	4	4	132.75	3	3	133.19	3	3	133.5
	3	9	9	126.41	8	8	127.24	8	8	127.54
fapp15	0	3	3	25.5	3	3	25.64	3	3	25.78
	1	5	5	35.38	5	5	35.62	5	5	35.94
	2	4	4	57.24	3	3	57.74	3	3	57.93
	3	3	3	25.26	3	3	25.45	3	3	25.72
	4	6	6	64.46	5	5	65.12	5	5	65.28
fapp16	0	4	3	1.16	4	3	1.57	4	3	1.77
	1	4	3	0.49	4	3	0.69	4	3	0.83
	2	4	3	2.14	4	3	2.55	4	3	2.7
	3	4	3	1.16	4	3	1.38	4	3	1.6
	4	5	5	1.26	5	5	3.83	5	5	4.16
	5	5	5	1.61	5	5	1.93	5	5	2.26
	6	6	6	1.6	5	5	2.68	5	5	3
	7	5	4	1.22	5	4	1.58	5	4	1.78
	8	5	4	1.23	5	4	1.68	5	4	2.06
	9	5	4	1.76	5	4	2.12	5	4	2.4
	10	5	4	2.2	5	4	2.49	5	4	2.78
fapp17	0	6	6	4.03	4	4	5.25	4	4	5.58
	1	5	5	7.44	4	4	8.65	4	4	8.92
	2	5	5	5.4	5	5	6.14	5	5	6.9
	3	4	4	8.54	4	4	9.92	4	4	10.46
fapp18	0	4	5	2.59	3	3	3.06	3	3	3.19
	1	5	6	2.6	5	6	2.93	5	4	3.47
	2	4	5	3.24	4	5	3.75	4	4	4.5
	3	4	4	2.59	4	4	2.75	4	4	2.92
	4	3	3	3.45	3	3	3.72	3	3	3.93
	5	4	4	2.57	4	4	2.73	4	4	2.9
	6	4	5	3.06	3	3	3.52	3	3	3.66
	7	13	14	5.64	9	10	6.83	7	6	8.06
fapp19	0	5	5	7.57	5	5	10.49	5	4	16.58
	1	5	5	7.83	5	5	14.43	5	4	20.5
	2	5	5	11.41	5	5	21.13	5	4	27.43
	3	5	5	23.82	5	5	29.4	5	4	34.68
	4	5	5	8.4	5	5	13.33	5	4	19.57
	5	11	16	13.26	7	8	30.29	6	5	36.66
fapp20	0	7	9	3.27	7	9	5.01	4	4	7.01
	1	3	3	3.17	3	3	3.71	3	3	4.26
	2	6	6	5.76	6	6	9.71	6	6	10.47
	3	4	5	10.68	4	5	11.17	4	4	12.34
	4	11	10	5.58	6	5	14.59	6	5	14.92
	5	15	18	4.05	8	8	10.73	6	6	12.98
	6	6	6	10.1	6	6	10.79	6	5	12.04
	7	6	6	12.93	6	6	13.45	6	6	14.19
	8	15	18	5.69	12	14	25.06	12	13	26.63
	9	21	29	3.52	21	29	4.95	9	8	12.42
fapp21	0	3	3	5.81	3	3	6.07	3	3	6.25
	1	3	3	5.69	3	3	5.86	3	3	6.04
	2	3	3	10	3	3	10.21	3	3	10.35
	3	3	3	10.11	3	3	10.32	3	3	10.46

Table 3.10: ROADEF2001 results

instance	level	IS V	IS C	IS T(sec)	IIS vars V	IIS vars C	IIS vars T(sec)	IIS ctrs V	IIS ctrs C	IIS ctrs T(sec)
fapp22	0	5	4	28.68	5	4	33.73	5	4	42.11
—	1	11	10	11.52	5	4	19.71	5	4	21.99
	2	15	13	15.31	11	10	87.15	11	10	145.41
	3	5	4	29.59	5	4	31.86	4	3	36.48
	4	12	9	10.54	6	5	15.24	4	3	21.63
	5	5	4	15.28	5	4	19.8	5	4	21.58
	6	9	8	37.75	7	6	47.05	7	6	51.91
fapp23	0	10	9	7.71	4	3	10.33	4	3	11.09
	1	15	14	6.98	4	3	8.41	4	3	9.17
	2	9	8	7.03	4	3	9.55	4	3	10.44
	3	7	6	10.29	4	3	12.87	4	3	13.64
	4	3	3	5.55	3	3	6.23	3	3	6.93
	5	3	3	6.75	3	3	7.43	3	3	8.13
	6	3	3	5.56	3	3	6.26	3	3	6.95
	7	3	3	5.55	3	3	6.26	3	3	6.95
	8	3	3	5.55	3	3	6.22	3	3	6.9
fapp24	0	6	5	64.01	6	5	80.72	6	5	80.99
	1	4	3	19.14	4	3	19.9	4	3	20.02
	2	4	3	31.38	4	3	32.57	4	3	32.69
	3	3	3	27.14	3	3	27.37	3	3	27.53
	4	3	3	26.82	3	3	27.14	3	3	27.29
	5	3	3	29.23	3	3	29.39	3	3	29.69
	6	6	6	40.68	6	6	41.35	6	6	42.33
fapp25	0	3	3	12.03	3	3	12.19	3	3	12.35
	1	3	3	12.06	3	3	12.2	3	3	12.44
	2	11	9	28.37	4	4	29.73	4	4	30
fapp26	0	4	4	35.93	4	4	36.16	4	4	36.41
	1	10	11	40.33	10	11	51.5	9	9	61.18
	2	8	7	45.49	6	6	46.74	6	6	47.25
	3	4	4	33.83	4	4	34.1	4	4	34.38
	4	8	6	176.98	4	4	180.29	4	4	181.83
	5	7	7	115.92	7	7	117.43	7	7	119.41
	6	5	5	87.58	4	4	99.01	4	4	100.95
fapp27	0	3	3	30.02	3	3	30.3	3	3	30.59
	1	5	5	31.84	4	4	32.78	4	4	33.17
	2	8	8	75.89	6	6	76.54	6	6	76.83
	3	18	17	133.65	6	6	134.91	5	5	135.4
	4	19	19	59.14	9	9	66.44	7	6	68.64
fapp28	0	4	3	144.65	4	3	145.37	4	3	146.11
	1	4	3	145.73	4	3	146.45	4	3	147.2
	2	4	3	157.96	4	3	158.68	4	3	159.56
fapp29	0	4	3	27.16	4	3	31.65	4	3	32.14
	1	5	4	17.71	5	4	18.89	5	4	19.57
	2	6	6	65.04	3	3	70.28	3	3	71.61
	3	6	5	22.69	6	5	23.67	4	3	24.94
	4	6	5	24.35	6	5	25.71	6	5	27.13
	5	5	4	28.68	5	4	29.62	4	3	30.87
fapp30	0	3	3	193.71	3	3	196.74	3	3	199.73
	1	3	3	193.69	3	3	196.83	3	3	200.33
	2	3	3	1016.52	3	3	1019.55	3	3	1022.62
	3	4	4	67.91	4	4	71.28	4	4	74.34

instance	level	IS V	IS C	IS T(sec)	IIS vars V	IIS vars C	IIS vars T(sec)	IIS ctrs V	IIS ctrs C	IIS ctrs T(sec)
fapp30	4	4	4	892.24	4	4	895.26	4	4	898.35
	5	3	3	312.45	3	3	315.44	3	3	318.5
	6	3	3	852.25	3	3	855.24	3	3	858.28
fapp31	0	4	3	40.21	4	3	45.24	4	3	50.4
	1	6	6	60.78	6	6	121.7	6	6	183.07
	2	5	5	231.51	5	5	301.41	5	5	371.32
	3	-	-	-	-	-	-	-	-	-
	4	-	-	-	-	-	-	-	-	-
fapp32	0	11	14	81.01	9	11	87.31	6	6	94.5
	1	8	11	159.47	3	3	162.47	3	3	162.79
	2	3	3	116.18	3	3	116.64	3	3	117.22
	3	7	9	75.67	7	9	77.09	5	5	80.7
	4	10	13	86.79	6	6	93.04	6	6	94.15
	5	11	14	100.64	7	8	106.79	7	8	107.43
fapp33	0	4	4	28.06	4	4	29.73	4	4	30.84
	1	9	10	34.05	9	10	41.38	6	5	44.3
	2	4	4	52.75	4	4	53.26	4	4	53.7
	3	10	10	38.36	7	7	45.15	7	7	49.57
	4	7	7	45.03	7	7	65.08	7	7	69.5
fapp34	0	4	5	19.31	4	5	20.47	4	4	24.15
	1	11	12	27.14	3	3	30.28	3	3	30.84
	2	4	4	9.82	3	3	11.22	3	3	12.24
	3	35	43	155.15	6	5	172	5	4	178.15
fapp35	0	5	5	48.38	3	3	48.98	3	3	49.26
	1	8	8	37.17	3	3	40.15	3	3	40.43
	2	6	6	39.22	4	4	40.1	4	4	40.47
	3	8	8	38.03	4	4	42.75	4	4	43.12
	4	11	12	59.12	3	3	66.47	3	3	66.75
	5	14	14	761.61	6	6	769.21	6	6	772.78
fapp36	0	4	3	18.64	4	3	18.94	4	3	19.1
	1	5	5	8.56	5	5	9.72	5	4	11.5
	2	13	13	63.86	8	7	216.56	8	7	217.1
	3	7	7	68.26	7	7	68.96	5	5	69.83
	4	5	5	43.34	3	3	44.82	3	3	45.2
	5	6	5	24.7	6	5	25.77	6	5	26.63
	6	6	6	37.08	5	5	38.19	5	5	38.73
fapp37	0	3	3	123.32	3	3	123.69	3	3	123.84
	1	3	3	104.08	3	3	104.22	3	3	104.36
	2	3	3	172.27	3	3	172.48	3	3	172.63
	3	10	9	784.08	3	3	785.81	3	3	786.78
	4	-	-	-	-	-	-	-	-	-
fapp38	0	6	5	247.83	4	3	252.21	4	3	252.6
	1	5	5	168.49	5	5	168.96	3	3	169.31
	2	11	12	365	8	9	378.54	8	9	381.06
fapp39	0	4	3	172.35	4	3	175.4	4	3	178.38
	1	6	5	290.13	6	5	293.65	6	5	298.86
	2	-	-	-	-	-	-	-	-	-
fapp40	0	6	5	96.39	6	5	99.59	6	5	104.29
	1	9	8	542.25	9	8	547.77	9	8	552.88
	2	9	8	528.23	5	4	549.58	5	4	555.7
	3	6	6	390.71	6	6	394.75	6	5	403.09

Table 3.11: ROADEF2001 results

The scenario fapp30 was a very particular case during our test; at level 2 (Arc Inconsistent level), it found a size of IS with 3 variables and 3 constraints while consuming about 1022.62 seconds on average. Since the level is not Arc-Consistent, *locator* can easily find a *deadend* variable with AC3. Most time is consumed in runtime in the construction and verification phases. With a detailed analysis, *constructor* finds a large critical constraints set on which *verificator* gives a satisfiability proof. This means that at the first construction procedure, the MRV strategy in *constructor* fails to find the right variable to form a critical constraints set. Despite this failure, the MRV strategy works well on all other instances.

In Figure 3.20, two curves show the sizes of the IS at different levels found in fapp05 and the execution times to obtain them. We observe that fapp05 is Arc-Consistent at level $8, 9$ and 10, and the computing time reaches the peak at 8 level. The computing time climbs according to the constraints levels, then decreases after the peak.

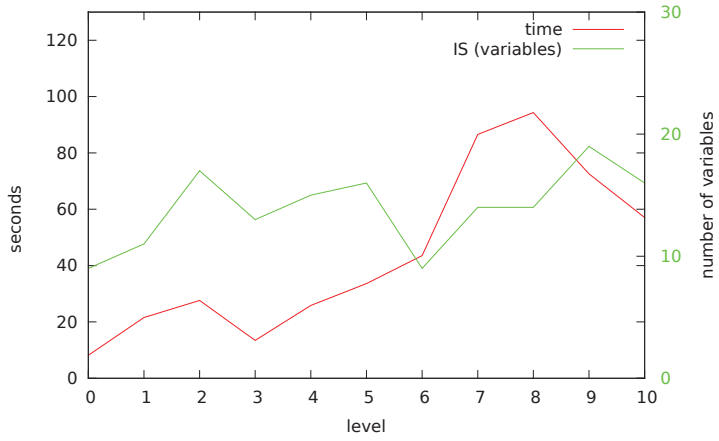

Fig. 3.20: Time and IS variable size on fapp05 for 10 levels

The computing time for the three components of *LCV* is relatively stable at different levels for fapp05 (see Figure 3.21). The *locator* occupies a large part of computational resource comparing to the other two. Inside the *locator*, the Arc-Consistency algorithm occupies almost 100% of the time when the instances is not Arc-Consistent up to level 7. From level 7 the instance is Arc-Consistent, the embedded local search is effective and constantly increasing the runtime percentage in *locator*. Therefore, we see that the problem properties highly influence the algorithm behavior, and hence the algorithm is self-adaptive intending good results for all cases.

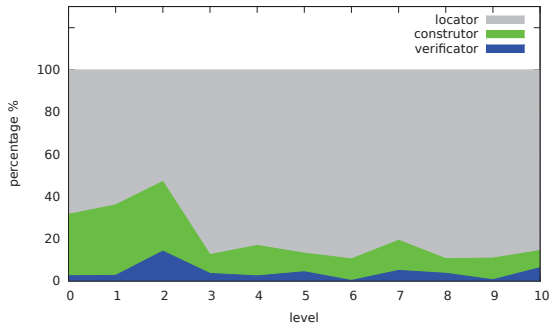

Fig. 3.21: Consuming time of the three components of *LCV* on fapp05

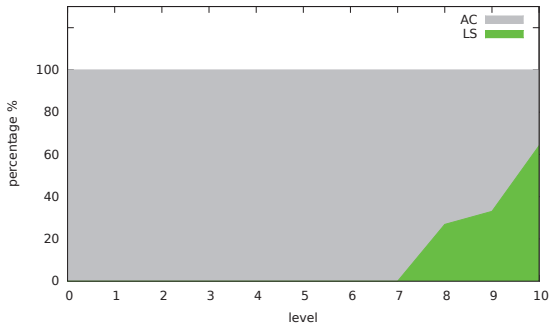

Fig. 3.22: Consuming time of Arc-Consistency and local search algorithms inside *locator* component on fapp05

3.6.3 Results for SOES

The performance comparison on the SOES benchmark is between our *LCV* algorithm and the in-house *SSA* algorithm (by DGA). *SSA* is an algorithm based on simulated annealing developed by the CELAR. We do not have more details about it, only its reference results are provided. The computational results are listed in Table 3.12, where $Vars.$ indicates the number of variables in the IIS found by both algorithms. The time limitation for both *LCV* and *SSA* algorithms was 6 hours. From the table, *LCV* finds always smaller IIS size than the one found by *SSA* algorithm.

scenario	SSA Vars.	*LCV* Vars.	scenario	SSA Vars.	*LCV* Vars.
SCEN 01	50	6	SCEN 11	770	30
SCEN 02	47	3	SCEN 12	702	5
SCEN 03	40	3	SCEN 13	182	29
SCEN 04	16	4	SCEN 14	300	11
SCEN 05	50	13	SCEN 15	1500	2
SCEN 06	38	5	SCEN 16	121	10
SCEN 07	50	4	SCEN 17	568	13
SCEN 08	50	6	SCEN 18	2000	2
SCEN 09	48	7	SCEN 19	154	39
SCEN 10	50	4	SCEN 20	249	13

Table 3.12: SOES results obtained by *SSA* and *LCV*

3.7 Conclusion

This chapter starts with a general introduction of the Frequency Assignment Problem (FAP), or more specifically, the Radio Links Frequency Assignment Problem (RLFAP) and the Frequency Assignment Problem with Polarization (FAPP). It is followed by detailed presentation of four related benchmarks - CELAR, GRAPH, ROADEF2001 and SOES. An initial algorithm, called the *2-phase* algorithm, to identify an IIS is presented and evaluated on the CELAR and GRAPH instances to aid the analysis both on the characteristics of instances and evaluated weak points of the algorithm design. From these preliminary results, two remarks surface:

1. The Arc-Consistency of instances needs to be verified as soon as possible, particularly in a FAP problem.
2. An IIS only represents a small part of the problem, the search need to focus exclusively on a small part of the problem.

As several instances of the benchmarks are not Arc-Consistent, the verification of such consistency is computationally cheap. When arc inconsistency is proven, the IIS is located immediately. The second remark is the key point to reduce the computing time on IIS identification. As is well known, the IIS identification in FAP instances is NP-hard. Based on the fact that an IIS is a connected component in the context of graph topology, it is wise to only work on relatively small subproblems instead of the entire problem.

Based on these experiences gained from the preliminary analysis, a general IS identifying routine *LCV* is introduced. The routine consists of three independent components - *locator*, *constructor* and *verificator*. The *locator* scans the entire problem and proposes a constraint which is potentially inside an IIS. Around such a constraint, the *constructor* forms a hard core by the means of connectivity. Finally, if the infeasibility of such a core is proven by the *verificator*, the core is an IS.

With the identified IS at hand, two subroutines called detectors to identify the IIS of variables and the IIS of constraints are detailed to detect an approximated IIS. First, an IIS of variables is identified on the proven IS. Then the IIS of constraints is extracted from the IIS of variables by the IIS of constraints subroutine.

The results of *LCV* on all benchmarks demonstrate the effectiveness and efficiency even in comparison with other approaches on CELAR/GRAPH and SOES problems. The general routine obtains smaller IIS and the speedup in computing time is considerable. The sizes of entire problems and the sizes of IIS have a huge impact on performance. The analysis on the results of ROADEF2001 shows that the *locator* is the most costly component among the three. The improvement in computing time can be focused on accelerating the Arc-Consistency algorithm and decreasing the consumption of embedded local search. On 5 runs, *LCV* found 100% success for CELAR/GRAPH, for 216 instances in ROADEF2001 and SOES. It failed to find IIS only on 4 ROADEF2001 instances.

Chapter 4

IIS in graph k-coloring problem

When you are looking at a map, there is perhaps a crucial question: *"How many colors do we need to color the countries of a map in such a way that adjacent countries are colored differently?"*. Such question demonstrates a classic example of the graph coloring problem. The Graph Coloring Problem (GCP) or more specifically in this chapter, the Vertex Coloring Problem, consists in coloring any two adjacent nodes with different colors. It is a famous and classic problem in operational research, the studies of such a problem and resolution techniques in the literature can be traced back to 19th century[1].

A graph G is not k-colorable if k is lower than G's chromatic number. In such a case, there exists at least one subgraph $G' \subseteq G$ which cannot be colored with k colors; if all strict subgraphs of G' are k-colorable, the subgraph G' is defined as a critical subgraph [78], which can be considered analogous of the IIS in the graph coloring problem.

This chapter is dedicated to identify the critical subgraphs in a graph coloring problem. It will be organized as follows: the definitions of the GCP and the IIS analogous to the GCP, the critical subgraph, will be described first. Several instances from the DIMACS benchmark will be illustrated after the definitions. Before entering the experimental results on critical subgraph detection, the approximated and exact approaches to solving the k-coloring problem will be summarized and two novel data structures to accelerate *TabuCol* will also be presented.

Finally, the *LCV* algorithm described in chapter 3 will be applied on critical subgraph identification in the k-coloring problem. The experimental results carried out by *LCV* on the DIMACS instances will also be compared with the results reported by Desrosiers et al. [78].

[1] http://en.wikipedia.org/wiki/Graph_coloring

4.1 IIS and critical subgraph

A graph $G = (V, E)$ consists of a set of nodes V and a set of edges E. The Graph Coloring Problem (GCP) consists in coloring any two adjacent nodes differently; such coloration is legal. Let S be the collection of all the independent sets of G; each independent set $s \in S$ contains a set of vertices that share no edges. Each independent set $s \in S$ contains a variable x_s whose value equals 1 if and only if the vertexes of s will be assigned the same color. Thus, the GCP can be formulated mathematically through the following model [100]:

$$\min \sum_{s \in S} x_s, \tag{4.1}$$

$$\text{s.t.} \sum_{s \in S: i \in s} x_s \geqslant 1, i \in V, \tag{4.2}$$

$$x_s \in \{0, 1\} \qquad s \in S. \tag{4.3}$$

The objective function (4.1) minimizes the number of independent sets (the number of colors respectively), the constraint (4.2) guarantees that each node in the graph belongs to at least one independent set. The last constraint (4.3) defines the variable x_s as a binary decision variable.

The intention of the GCP is to find the minimal k number of colors which allows us to color all adjacent nodes differently. Such a minimal number, k, for a graph G is called the chromatic number of G, denoted $\chi(G)$. The determination of $\chi(G)$ is NP-hard [101]. A graph G is k-colorable if $\chi(G) \leq k$. If a graph G is k-colorable, then all its nodes can be divided into k independent sets.

Definition 29 (k-coloring). Given a graph $G = (V, E)$ and a positive integer k such that $k < |V|$, the graph k-coloring problem is to determine whether there exists a legal vertex coloring using k colors.

The k-coloring problem can be conveniently formulated as a CSP. The nodes are the variables of the CSP, the edges between them form the constraints of the problem and the domain of the problem consists of the given k colors.

In case the given k is lower than the chromatic number $\chi(G)$, the graph G is not k-colorable. It raises the same question from the previous chapter: can we identify an induced subgraph G' which is not k-colorable, while all its strict subgraphs are k-colorable? Such a subgraph G' is named the critical subgraph:

Definition 30 (Critical Subgraph). A critical subgraph of graph G is an induced subgraph $G_{critical} \subseteq G$, which cannot be colored with k colors. It becomes k-colorable iff any of its nodes are removed.

The critical subgraph has several important characteristics. Firstly, it is an induced subgraph:

Definition 31 (Induced Subgraph). An induced subgraph $G' = (V', E')$ of a graph G is such a subgraph that G' contains all the edges $(x, y) \in E$ with nodes $x, y \in V'$.

Considering the definition of an IIS in a CSP, the critical subgraph is an IIS of variables in a CSP. Secondly, it is also a connected component of a graph G:

Definition 32 (Component). A maximal connected subgraph $G' = (V', E')$ of G is called a component (also named connected component) of G [102].

A subgraph which is a component also addresses the connectivity property of an IIS in a CSP. One important application of finding critical subgraph in [102] is to determine the chromatic number of a given graph G. Since the complexity of determining the chromatic number of the graph is exponential, if a small critical subgraph can be identified, then the cost of verification on such a small size critical subgraph by a complete search is affordable.

In next section, the benchmark used to examine the performance will be illustrated and explained. All these instances are from the well known DIMACS benchmark [103].

4.2 DIMACS instances

The DIMACS GCP instances is a set of instances proposed by different authors, which are dedicated to examine the performance of algorithms. Regarding the range of results on this benchmark, it is an excellent benchmark to examine critical subgraph identification algorithms. The instances files can be found at Carnegie Mellon University[2]. There are several categories of graph which are illustrated as followings.

Book Graphs. This benchmark is based on the relations among the characters of several classic novels which include Tolstoy's Anna Karenina (anna.col), Dicken's David Copperfield (david.col), Homer's Iliad (homer.col), Twain's Huckleberry Finn (huck.col), and Hugo's Les Misérables (jean.col). In Figure 4.1, the graph shows the topology of anna.col instance. Notice that there are duplicated edges in the instances.

Queen Graphs (queen*_*.col). Given an $n \times n$ chessboard, a queen graph is a graph with n^2 nodes, each corresponding to a cell of the board. Two nodes are connected by an edge if the corresponding cells are in the same row, column, or diagonal. Unlike some of the other graphs, the coloring problem on this graph has a natural interpretation: given such a chessboard, is it possible to place n queens on the board such that two queens are not in the same row, column, or diagonal? Figure 4.2 shows the graph topology of these Queen instances.

[2] http://mat.gsia.cmu.edu/COLOR/instances.html

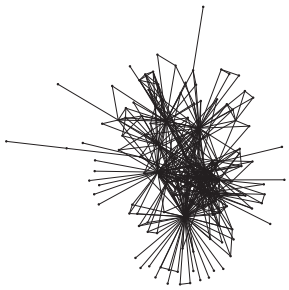

Fig. 4.1: anna.col (undirected without duplicated edges)

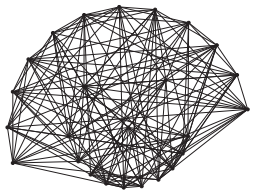

Fig. 4.2: queen5_5.col

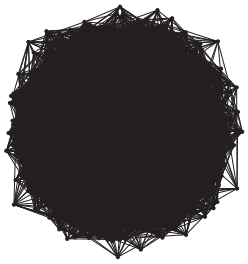

Fig. 4.3: le450_5a.col

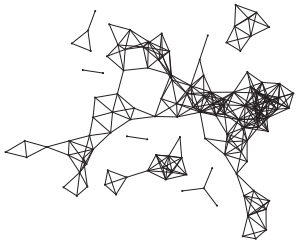

Fig. 4.4: miles250.col

Leighton Graphs (le*.col). These instances were generated randomly with a density lower than 0.25. The chromatic number of each instance is fixed. Figure 4.3 shows one graph topology, the instance has a very high density, and each node is over-connected.

Game Graphs (game120.col). The games played in a college football season can be represented by a graph (see Figure4.6) where the nodes represent each college team. Two teams are connected by an edge if they played each other during the season. Knuth gives the graph for the 1990 college football season.

Miles Graphs (miles*.col). These graphs are similar to geometric graphs. The nodes represent a set of United States cities and two nodes are connected if the cities are close enough on the basis of the distance between them given by road mileage in 1947. Figure 4.4 shows that this instance is well structured and has several independent components.

Register Graphs (fpsol*.col, inithx.i.*.col, mulsol.i.*.col, zeroin.i.*.col). These graphs come from the problem based on register allocation for variables in real codes. The graph topology shown in Figure 4.5 indicates that such instance has two subsets of nodes heavily connected.

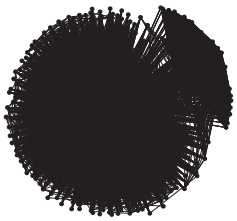

Fig. 4.5: fpsol2.i.2.col

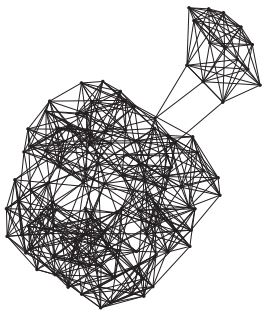

Fig. 4.6: games120.col

Beside above illustrated instances, there are also instances proposed with different purposes, including random generated instances (DSJC*.col), geometric random graphs (DSJR*.col, r*.col), Quasi random graphs (flat*.col), etc. The readers can refer the paper of Chiarandini et al. [104] or to the site of Carnegie Mellon University for further information.

By referring to the above instances, the random generated instances are unstructured with high density, while the real or semi-real world instances are well structured with particular graph topology. In the next section, several k-coloring problem resolution techniques are briefly reviewed as k-coloring is not the main topic of this thesis.

4.3 Solution techniques for k-coloring problem

The k-coloring problem is a well studied classic combinatorial optimization problem. Many heuristic and exact solution techniques have been proposed in the literature. The challenge of finding more efficient and effective algorithms always attracts researchers in the operational research community. The numerous methods dealing with k-coloring problem can be roughly classified into two major categories, heuristic and exact approaches.

Recently Galinier and Hertz [87] and Malaguti and Toth [105] give a comprehensive survey of the solution techniques for the GCP. Also Chiarandini and Gualandi maintain an excellent website concerning the solution techniques of the graph coloring problem[3]. Interested readers can refer to these materials for detailed information.

[3] http://www.imada.sdu.dk/∼marco/gcp/

4.3.1 Heuristic approaches

Due to the NP-complete nature and the size of real world instances, many solution techniques of the k-coloring problem have been proposed under the umbrella of heuristic methods. As an approximated approach, an heuristic cannot guarantee the optimal solution since only a partial search space of the problem is explored and it may be trapped in *local optima*.

The breakthrough of heuristic design for the GCP begins with the famous *DSATUR* algorithm proposed by Brélaz [46] in 1979. In 1987, *TabuCol* [97] achieves a significant performance gain despite its simplicity. In 1999, the hybrid evolutionary algorithm *HCA* by Galinier and Hao [106] gives another performance increase thanks to its elegant crossover design and the replacement of mutation by a local search operator.

These algorithms illustrate the development from simple heuristics, to local search meta-heuristics and then to evolutionary meta-heuristics. In the following sections, several approximation algorithms will be explained based on such a classification.

4.3.1.1 Simple heuristics

DSATUR's success is based on the use of saturation degree which is similar to the Minimal Remaining Value (MRV) heuristic [107]. Even combined with a simple descent procedure it obtains good results [46]. Recently, it is considered as a quick heuristic method to generate the initial solutions for meta-heuristics [108]. In contrast to the strategy adopted by *DSatur*, Leighton's *RLF* (Recursive Largest First) [109] firstly colors the maximal independent set and has better average performance than *DSatur* [110]. Bollobás and Thomason [111] also address the importance of the maximal independent set of uncolored nodes by means of a backtrack algorithm. Interested readers can refer the surveys [87, 105] for more information.

4.3.1.2 Stochastic local search

Stochastic local search begins with an initial solution and iteratively moves from a candidate solution in the search space to another neighboring solution. The search space may contain partial candidate solutions (in construction methods) or complete candidate solution (in repair methods), the move decision is based on a limited amount of local information [112]. In this section, several well known algorithms under the category of stochastic local search will be explained.

Focusing on the *Min-Conflict* strategy, *TabuCol* adopts a very simple 1-move (or 1-exchange) neighborhood structure. The neighbor solutions are generated by only changing the color of one node among all the conflict nodes. Using a tabu thresholding technique and a tabu list, the method achieves better performance than the *Simulated Annealing* approach [113, 114]. Algorithm 14 illustrates the routine of *TabuCol*. Its elegant critical 1-move neighborhood structure reduces the number of neighborhood solutions, while at the same time, good solution quality is kept.

Algorithm 14: *TabuCol*

Input : a graph $G = (V, E)$ and k number of colors
Parameters: $MaxIter$, L and λ
Output : a complete assignment $\mathcal{A}^*$ on G with k colors

Create a random assignment $\mathcal{A}$;
$\mathcal{A}^* \leftarrow \mathcal{A}$ and $iter = 0$;
$TabuList \leftarrow \emptyset$;
repeat
    $iter \leftarrow iter + 1$;
    Choose a candidate 1-move (x_i, a_i) among all conflict variables with minimum violation;
    Add move $(x_i, \mathcal{A}(x_i))$ into $TabuList$ for $L + \lambda \times Conflict(\mathcal{A})$ iterations;
    $\mathcal{A} \leftarrow \mathcal{A} + (x_i, a_i)$;
    if $f(\mathcal{A}) < f(\mathcal{A}^*)$ **then** $\mathcal{A}^* \leftarrow \mathcal{A}$
until $f(s) = 0$ *or* $iter = MaxIter$;

PartialCol, proposed by Blöchliger and Zufferey [115], extends the solution from a k-legal partial solution. It also adopts a reactive tabu tenure *Foo-scheme* which adaptively corresponds to the fluctuation of the objective function. Such a scheme avoids the unpromising part of the search space and helps the tabu tenure to escape from these areas. However experimental results show that the construction method is only better on *flat* instances than the repair method of *TabuCol*.

Avanthay et al. [116] adopt a *VNS* (Variable Neighborhood Search) [117] approach to solve the GCP, they propose various neighborhood structures to diversify the neighborhood solutions and to explore different search spaces. Hertz et al. propose the *VSS* (Variable Space Search) [118] algorithm which, not only switches among the neighborhood structures, but also defines different objective function for each neighborhood structure. The algorithm uses cyclic executions of three different heuristics which is derived from Formulation Space Search (FSS) [119]; also it is similar to *Hyper-heuristics* introduced in [120]. Chiarandini and Stuetzle simplified *VNS* and proposed an *ILS* (Iterated Local Search) [121] approach to solve the k-coloring problem. Devarenne et al. design a mechanism to avoid the local optima by detecting the loop of visited nodes [122]. Dib et al. propose *TabuNG* [123] to reduce the search space by recording the partial solution which cannot lead to the feasible solutions.

In these approaches, different neighborhood structures are proposed to diversify the neighborhood solutions and to exploit the different search space avoiding the local optima. In [124] and [116], the authors have proposed various neighborhood structures for the k-coloring problem.

The stochastic local searches for k-coloring can be roughly classified into two categories: construction and repair algorithms. The first consists in extending a partial consistent solution to a complete one, the latter attempts to repair the current inconsistent or non-optimal solution to a consistent or improved solution. Recently, the hybridization of these two manners was also adopted. Table 4.1 proposes a classification of algorithms mentioned in this section.

Heuristics	Construction	Repair	Hybrid
[97]*TabuCol*		x	
[115]*PartialCol*	x		
[116]*VNS*		x	
[118]*VSS*			x
[121]*ILS*	x		
[122]*ALS*		x	
[123]*TabuNG*	x		

Table 4.1: Category of some local search algorithms

In [87], Galinier and Hertz point out that stochastic local search is very efficient to solve graph coloring problem. The experimental results shown in [104] compare several stochastic local search algorithms, the authors conclude that *TabuCol* is robust and efficient despite its simplicity. Several algorithms surpass its performance but with higher runtime or complicated algorithm designs. However such advantages are often visible for a subset of specific instances. As suggested in [106], *TabuCol* is an ideal local search algorithm to integrate into other search algorithms thanks to its elegant design and efficiency.

4.3.1.3 Evolutionary algorithms

In 1975, Holland proposed a nature inspired meta-heuristic, called the genetic algorithm [125]. Then, a specific category of algorithms named Evolutionary Algorithm derived from these ideas has been developed. These algorithms adopt both individual and population evolutions and natural selection and are applied to many optimization problems [126].

Galinier and Hao proposed *HCA* (Hybrid Coloring Algorithm) [106] which achieves better solution quality than the simple adoption of a genetic algorithm on GCP [127]. *HCA* replaces the original mutation operator by *TabuCol* and uses an elegant crossover operator. The *Greedy Partitioning Crossover* operator consists in selecting the maximal cardinality color partition ignoring nodes already chosen from other parent. Such an offspring strategy creates one individual with high stable independent sets from both parents.

Figure 4.7 illustrates three steps of crossover in *HCA*. In each generation of *HCA*, two individuals $P1$ and $P2$ are selected from the population and considered as parents to form a new individual $Child$. Initially, the child inherits the nodes in the biggest color partition from $P1$. Then, the duplicated nodes in $P2$ are removed (the shaded parts in $P2$) and the largest next color partition is selected in $P2$, it is used as the second color partition of the child. Finally, another partition of $P2$ is selected for the child. The procedure stops when the child has k independent sets and the nodes not yet selected will be assigned randomly. After the crossover operator, the *TabuCol* algorithm is used to mutate the child individual.

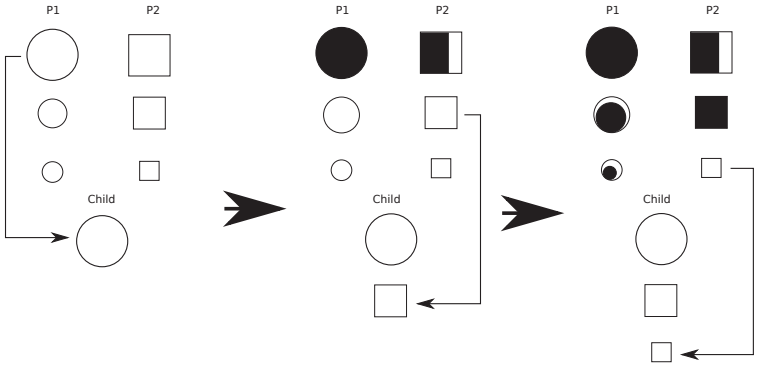

Fig. 4.7: Crossover in *HCA*

Glass and Pruegel-Bennett [128] modify *HCA* by replacing *TabuCol* with a less effective heuristic search (Vertex Descent Algorithm) and by extending the size of the population. Their experimental results indicate that high quality local search is more effective than the classical mutation operator.

Regarding the success of *HCA* even with small size population, *AMACOL* (Adaptive Memory Algorithm for the k-COLouring) [129] is another algorithm which employs a central adaptive memory structure [130] to record parts of solutions instead of the entire individual. The child is generated by processing a specific recombination operator which respects the maximal stable sets from the central memory. Then the central adaptive memory updates itself with the maximal stable sets of the child individual. The maximal stable set is generated by removing the conflict nodes

and adding the non-adjacent nodes which share the maximal number of common neighbors with the nodes inside the stable set.

MMT (Evolutionary Algorithm and Column Optimization) proposed by Malaguti et al. [131] adopts a two-phase evolutionary approach. In the first phase, *HCA* is used to find a solution based on pre-defined lower and upper bounds. In order to improve the results, the second phase, the independent sets found and recorded during the execution of *HCA* are used by a Lagrangian heuristic algorithm, CFT, proposed by Caprara et al. [132] under the set covering formulation.

Lu and Hao [108] proposed a memetic algorithm called *MACOL* (Memetic Algorithm for k-COLoring) which combines a genetic algorithm with a tabu search procedure. They adopt a partition crossover instead of an assignment crossover. A multi-parents crossover operator *AMPaX* improves the performance of the 2-parents crossover operator [106], and shows that choosing a random number of parents provides more diversification in difficult instances. The evaluation function of their approach consists in computing the number of conflict inside color partition.

Porumbel et al. [133] mention the impact of the population diversity on evolutionary algorithm performance. They define a measurement distance to indicate the similarity between two individuals. By rejecting the similar individual, diversity in population is guaranteed. Their experimental results show a good performance improvement by comparison with *MACOL* and *MMT*. They conclude that the distance between individuals in the population has more impact on algorithm performance than the number of individuals.

Figure 4.8 illustrates the loose relation among several evolutionary algorithms for the k-coloring problem. The improvement of solution quality carried out by evolutionary algorithms is visible by comparing with stochastic local search algorithms. While at same time, the evolutionary approach often causes computational overhead.

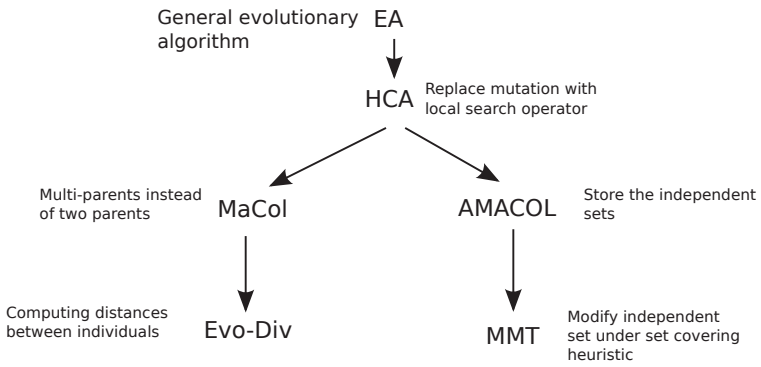

Fig. 4.8: Several evolutionary algorithms for k-coloring

4.3.1.4 Summary of heuristic methods

From the literature, the best solution quality is found using evolutionary algorithms as opposed to the stochastic search algorithms. While the simple heuristics and several stochastic local search algorithms obtain acceptable solution quality within shorter computational time, their solution quality on graph coloring instances based on real world applications are quite good considering the runtime used, but poor performance is found on random or quasi-random instances on which the quality of evolutionary approach is better.

Global intensification and diversification are two key cornerstones in the meta-heuristic process. Well dedicated neighborhood structures may efficiently solve many real world problems thanks to the understanding of specific problem structures, but when facing academic problem instances, a more sophisticated diversity strategy is needed to avoid local optima.

4.3.2 Exact approaches

Numerous exact algorithms dealing with the GCP have been proposed, such as Brown's implicit enumeration algorithm [134], Brélaz's modification based on the Brown's algorithm, Sewell's improvement [135] of Brélaz's approach, Peemöller's correction [136] on Brélaz's modification, etc.

A Branch and Price column generation approach is proposed by Mehrotra and Trick [100] which is based on the *Set Covering* model (see Equation (4.1)). The experimental results of such an approach show that it can solve the random instances up to 70 vertices and the random geometric graphs up to 250 vertices. Hansen et al. adopt a *Set Packing* model [137] and show an improvement on solving larger instances up to 80 vertices. Méndez-Díaz and Zabala improved Hansen et al.'s model by using the symmetry property of the model.

These methods have been proven to work well on small size instances [138, 139]. Herrmann and Hertz [138] proposed an hybrid approach which significantly improves the algorithm effectiveness on relatively larger instances. The algorithm can be essentially expressed as Algorithm 15, it is based on a very simple property:

Property 3. In GCP, if a subgraph $G' \subseteq G$ has the same upper bound as G, then such a subgraph G' has the same chromatic number as G, noted $\chi(G') = \chi(G)$.

Based on Property 3, the original graph G is reduced by iteratively removing the nodes which do not change its upper bound. If removing a node changes the upper bound, the removal will be reversed (Line 4 of Algorithm 15). Eventually, the algorithm will obtain a subgraph G', whose upper bound will be changed by removing any of its

node. Then the smaller subgraph G' will be verified by a branch and bound algorithm proposed by Peemöller [136] ($ChomaticNumber()$ of Algorithm 15) which is a correction of Brélaz's modification of Brown's algorithm [46]. Even if the complexity of Peemöller's approach is exponential, thanks to the reduced size of G', the Herrmann and Hertz's approach has proven its effectiveness on larger instances than the previous exact methods in the literature. Desrosiers et al. [78] adopt Herrmann and Hertz's algorithm to verify the chromatic number of identified critical subgraphs.

Algorithm 15: Simplified version of [138] without the correction procedure

input : a graph $G = (V, E)$
output: the chromatic number $\chi(G)$ of graph G
1 $G' \leftarrow G, \mu \leftarrow UpperBound(G)$;
/* Reduce the graph G */
2 **foreach** $v \in V$ **do**
3 $G' \leftarrow G' - \{v\}$;
4 **if** $UpperBound(G') \neq \mu$ **then** $G' \leftarrow G' \cup \{v\}$;
5 **end**
/* Determine the chromatic number */
6 **return** $\chi(G) \leftarrow ChomaticNumber(G')$;

4.4 Novel data structures to speed up *TabuCol*

As we use *TabuCol* in the constructor routine of our *LCV* algorithm for IIS identification in FAP and graph k-coloring problem, in this section we present two novel data structures to improve the performance of *TabuCol*. *TabuCol* can be divided into two main procedures – move evaluation and move update. Based on its *1-move* neighborhood structure, the move evaluation is run for all colors on all violated nodes, which may occur thousands of times per iteration. Comparing with the evaluation procedure, the move update procedure is run once per iteration on only one node. Acceleration of *TabuCol* can be achieved by reducing the time consumption on the move evaluation.

Our proposal is to store the states of inconsistency, called the *Tabling Strategy*, on each color per node. Instead of an exponential comparison procedure, a linear search algorithm can find the minimal inconsistency color. The first structure, called gamma table, is applied for FAP and graph k-coloring problem, while the second structure, called bounds list, is an additional structure and is very efficient for T-coloration constraints such as FAP constraints.

4.4.1 Gamma table for FAP and graph k-coloring

In [87], Galinier and Hertz mentioned a $n \times k$ centralized γ-matrix to represent the state and level of inconsistency for each color per node, where n is number of nodes and k is the number of given colors. Thanks to its elegant design, a significant acceleration was achieved on the performance of *TabuCol*. Slightly different from the data structure mentioned in [87], we propose a decentralized version of γ-matrix, called gamma table, to accelerate *TabuCol* on solving both the FAP and the graph k-coloring problem. Example 4 gives an illustration of gamma table on the graph k-coloring problem.

Example 4. Given 5 adjacent nodes A, B, C, D, E with connections as in Figure 4.9. Let the color assignment of four nodes be $\{A : Red, B : Blue, C : Blue, D : Yellow\}$ and the current color on node E is Red, so we have one violation.

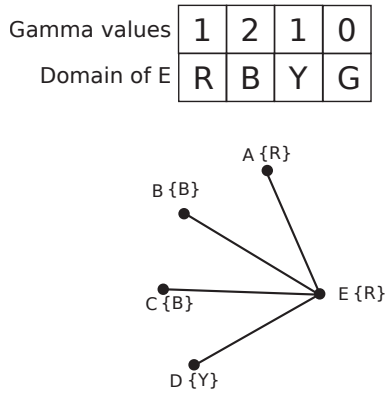

Fig. 4.9: Example 4 – connections and gamma table for node E

Figure 4.9 shows the gamma table of node E in Example 4. Based on the color assignment of neighboring nodes of E, the inconsistency states of colors in the domain of E are: Red on one edge, $Blue$ on two edges and $Yellow$ on one edge. The fourth color $Green$ is consistent with the neighborhood assignment of E. In order to store this information and eliminate the constraint verification, the inconsistency state can be stored in a table named the *gamma table* where the gamma values are the number of inconsistencies per color per node.

Suppose now that we have such gamma table for each node. Using a linear search in the gamma table, we know that $Green$ is the best move for E since $Green$ has the minimum inconsistency state in the domain of E. If the move is accepted by *TabuCol*, the assignment on E is changed, so the gamma tables on E's neighbors $\{A, B, C, D\}$ need to be updated. All gamma values of color $Green$ in the domains of $\{A, B, C, D\}$ will be increased with one unit which

indicates the inconsistency of this color with node E and all the gamma values of Red are decreased with one unit. Also the value of the objective function will be reduced with the difference between the two gamma values of E for $Green$ and Red.

The time complexity of the move evaluation and the move update procedures of our proposal and the original *TabuCol* data structure designed by Hertz and Werra [97] is shown in Table 4.2, where e is the maximum degree of the graph and k is the given number of colors. The space complexity to implement such a structure is $\mathcal{O}(nk)$, where n is the total number of nodes. Comparing with [97], the complexity of move evaluation for each node per iteration is significantly reduced. Despite the higher complexity of the move update with gamma table, we have a significant runtime performance gain on *TabuCol* because during each iteration of *TabuCol*, the number of execution of the move evaluation is much larger than the move update's. Figure 4.10 shows the time variation according to the number of *TabuCol* iterations on one DIMACS instance. The gamma table only consumes 38.3% of time in comparison with [97] to complete the same task.

Complexity per node	Hertz and Werra [97]	Gamma table
Move evaluation	$\mathcal{O}(ek)$	$\mathcal{O}(k)$
Move update	$\mathcal{O}(1)$	$\mathcal{O}(ek)$

Table 4.2: Comparison for move evaluation and update between Hertz and Werra [97] and gamma table

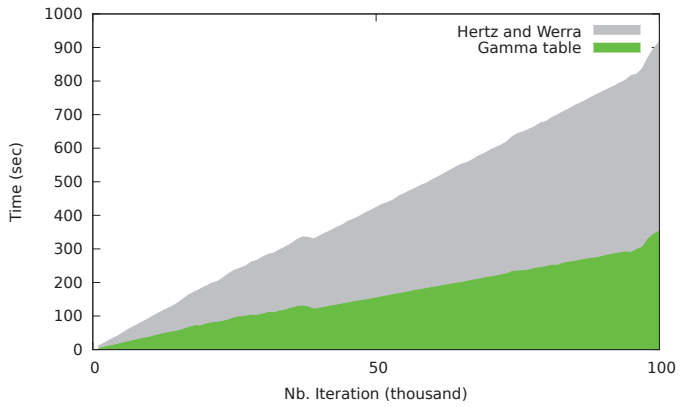

Fig. 4.10: Time comparison on DIMACS le450_5a with 4 colors

Our implementation only decentralizes the γ-matrix onto the node level. It shows no particular benefit in solving the k-coloring problem, but in FAP instances such decentralized data structure can reduce the space complexity since the domain sizes are various for each node.

4.4.2 Bounds list for FAP

In addition to the decentralized gamma table, we propose another data structure for *TabuCol*, named *bounds list*, to deal with FAP instances. Each bound in the bounds list consists of a bound value and a gamma value. Such a data structure benefits from the properties of the FAP binary constraints which are:

$$|f_i - f_j| > \delta_{ij} \Rightarrow \begin{cases} Eq_a : f_j < f_i - \delta_{ij} \\ Eq_b : f_j > f_i + \delta_{ij} \end{cases} \tag{4.4}$$

Equations Eq_a and Eq_b separate the domain of variable j into three parts: the values lower than $f_i - \delta_{ij}$, the values greater than $f_i + \delta_{ij}$ and the values in the interval $[f_i - \delta_{ij}, f_i + \delta_{ij}[$. Equation (4.4) creates two bound values $(f_i - \delta_{ij})$ and $(f_i + \delta_{ij})$ that we put in a bounds list data structure. Thus the bounds list consists of two bound values bounding the gamma values.

The difference between the gamma values in the bounds list for FAP and the gamma values in gamma table for FAP is that in the bounds list the gamma values have plus and minus $(+/-)$ signs to calculate the inconsistency state of the values in the variable domain. Let us consider the following example:

Example 5. Let's define a duplex communication with two radio links f_1 and f_2, the available frequencies for these two links are $\{2, 4, 6, 8, 10, 12, 14\}$ and the constraint to avoid interference is defined as $|f_1 - f_2| > 3$.

Firstly we sort the domain values from smaller to larger. If the link f_1 is assigned frequency 6, then two bound values of 3 and 9 are created for f_2. The domain of f_2 is separated by these two bounds into three intervals $\{2\}$, $\{4, 6, 8\}$ and $\{10, 12, 14\}$ (see Figure 4.11). The gamma values associated with the two bound values are $+1$ and -1. Initialized with a zero inconsistency state, all values greater than bound 3 have the inconsistency state of 1 (obtained by $0 + 1$); all values greater than bound 9 have the inconsistency state of 0 (obtained by $0 + 1 - 1$). It is noticed that the inconsistency state of each value a in the domain equals the sum of all gamma values associated with bound values smaller than a.

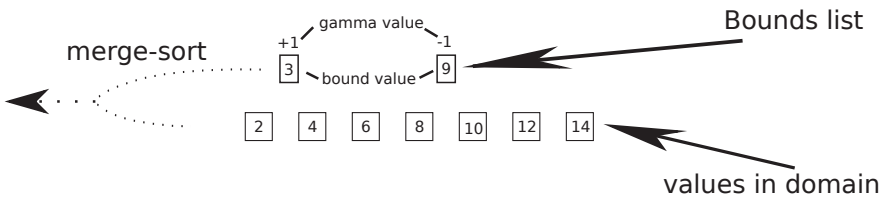

Fig. 4.11: Bounds list of Example 5

In order to calculate the inconsistency state in such a manner, the bounds and values of the domain need to be sorted in the same way. With the ordered bounds list and domain values, a merge sort is used to rapidly determine the inconsistency states of all domain values. The space complexity of using a bounds list is $\mathcal{O}(mt)$, where m is the total number of constraints and t is the upper number of bound values created for all constraints. Let d be the maximal cardinality of the domains, the time complexity to evaluate all the values of each variable (move evaluation in *TabuCol*) is $\mathcal{O}((t+d)\log(t+d))$. Such complexity is obtained by the usage of *Merge Sort*. The time complexity of using a bounds list for move update for the FAP is $\mathcal{O}(e)$, where e is the maximal number of constraints involved in one variable; it is lower than the time complexity of using a gamma table (see Table 4.2).

4.4.3 Summary of novel data structures

Table 4.3 shows the evaluation and update time complexity and the space complexity for both novel data structures, where e is the maximum degree of the graph, d is the total number of domain values, n is the number of nodes, m is the total number of constraints, and t is the upper number of bound values created for all constraints. The choice between the two approaches for the FAP problem, and more generally for the CSP with binary constraints, depends on the number of bounds created from the constraints: if the number of bounds is small, it is worth choosing the bounds list approach, otherwise, the gamma table is better because of its space and time complexity.

	Gamma table	Bounds list
Time complexity(move evaluation)	$\mathcal{O}(d)$	$\mathcal{O}((t+d)log(t+d))$
Time complexity(move update)	$\mathcal{O}(ed)$	$\mathcal{O}(e)$
Space complexity	$\mathcal{O}(nd)$	$\mathcal{O}(mt)$

Table 4.3: Time and space complexity for both novel data structures

4.5 Experimental results on critical subgraph identification

In the previous chapter, we introduced a general routine to detect an IIS in the FAP named *LCV*. Thanks to the similarity between the FAP and the k-coloring problem, we intend to apply *LCV* on DIMACS instances with minor modifications. At first we present the state of the art of the methods in critical subgraph identification in the k-coloring problem.

Desrosiers et al. propose three algorithms to identify the critical subgraph [78], *Removal*, *Insertion* and *Hittingset*. Among these three algorithms, *Insertion* shows the best performance. The authors also show that *Insertion* with preprocessing of the *prefiltering* algorithm can achieve better performance than only using *Insertion*. Let's first recall the *Insertion* algorithm (see Algorithm 16).

Algorithm 16: Insertion [78]

Input : a set of constraints C
Output: a subset of constraints $H \subseteq C$ forming an IS

```
1  H ← ∅, C_v ← ∅;
2  repeat
3      if C_v ≠ ∅ then
4          C ← C \ C_v;
5          choose one constraint c ∈ C_v;
6          H ← H ∪ {c};
7      end
8      C_v ← MinConflict(C, H);
9      if C_v = ∅ then return Fail to find IIS;
10 until H ∩ C_v ≠ ∅ ;
```

In the *MinConflict*() (Algorithm 16 Line 8), the satisfaction of the constraints in H is done by introducing all constraints of H with significant heavy weights in its objective function. The objective of *MinConflict* is to minimize the sum of the weights on constraints:

$$\min \sum_{c \in C} w(c) \times x(c) \tag{4.5}$$

$$w(c) \in \{1, MAX\} \tag{4.6}$$

$$\text{where } x(c) = 1 \text{ if } c \text{ is violated}, x(c) = 0 \text{ if } c \text{ is satisfied} \tag{4.7}$$

By minimizing Equation (4.5), the heavily weighted constraints are satisfied. *MinConflict*() is run iteratively until at least one edge cannot be satisfied in H. *TabuCol* is adopted as the embedded local search in *MinConflict*(). Considering Line 4 of Algorithm 16, it removes all the violated edges except the chosen constraint c. Thus the subproblem entering *MinConflict*() will contain only one violated edge. Since *TabuCol* exclusively changes the color on violated nodes, the edges with zero weight will be ignored and not be evaluated. That is the main reason why the *Insertion* algorithm works very rapidly.

It is worth mentioning that the *prefiltering* algorithm (Algorithm 17) is used to reduce the size of the graph in the first step. From the authors' observation, it dramatically accelerates the critical subgraph identification by providing a smaller size graph for the *Insertion* algorithm, denoted H.

Algorithm 17: prefiltering [78]

input : a set of constraints C
output: a subset of constraints $H \subseteq C$ forming an IS

```
1 H ← ∅, C_v ← ∅;
2 repeat
3     if C_v ≠ ∅ then
4         H ← H ∪ {C_v};
5         C_v ← ∅;
6     end
7     C_v ← MinConflict(C, H);
8 until H ∩ C_v ≠ ∅ ;
```

From the pseudo-code of *prefiltering*, the algorithm stops when at least one critical subgraph is included in the output subgraph. At the same time, it destroys any critical subgraphs which have a larger size than the included one.

Table 4.4 shows the comparison between Desrosiers et al. *Insertion+prefiltering* approach and *LCV* approach. The results of *Insertion* is obtained on using a computer with a Athlon 1.6Ghz CPU and 512Mb of RAM [78], the results of *LCV*'s are carried out on a PC Intel Core 2 Duo 2.6Ghz with 3.8Gb RAM. *LCV* is implemented in C++ without parallelization.

The k column indicates the number of colors available for each instance, and it is fixed such that each instance is not k-colorable. The second, third and fourth columns list the sizes of the critical subgraphs and the minimum computational time in 5 runs obtained by *Insertion+prefiltering*. The fifth and sixth columns give the size of critical subgraphs found by *LCV*. The $Success/5$ column indicates the number of successes in 5 executions of *LCV* to get the same critical subgraph size as *Insertion+prefiltering*. The last column indicates the minimum execution time in seconds when the *LCV* algorithm is successful, without the execution time of the *verificator* routine.

From the comparison table, the results of *LCV* are notably worse than the ones carried out by *Insertion* in [78] for the sizes of critical subgraphs and the computational times. There are 14 instances out of 37 for which *LCV* obtains larger sizes than *Insertion*. For the rest of the instances, *LCV* finds the same size of critical subgraphs but with a longer runtime and a lower success rate in 5 runs.

To explain the difference in the results, it is noticed that the collaboration between *Insertion* and *TabuCol* is a perfect match for k-coloring. First, an effective heuristic such as *TabuCol*, may only violate a minor number of nodes in each critical subgraph. With the help of an embedded weighting system [78], the heuristic may find the critical constraints

Instance	k	*Insertion+prefiltering* [78]			*LCV*			
		$\|V\|$	$\|C\|$	Min Time(sec)	$\|V\|$	$\|C\|$	$Success/5$	Min Time(sec)
le450_5a	4	5	10	10.7	5	10	1	28.65
le450_5b	4	5	10	13.4	5	10	1	28.71
le450_5c	4	5	10	17.8	5	10	1	83.4
le450_5d	4	5	10	16.7	5	10	2	27.96
anna	10	11	55	0.3	11	55	2	1.09
david	10	11	55	0.3	11	55	1	3.15
homer	12	13	78	0.8	30	208	–	9.96
huck	10	11	55	0.2	11	55	5	5.48
jean	9	10	45	14.2	10	45	5	0.14
games120	8	9	36	0.4	9	36	5	0.57
miles250	7	8	28	0.2	8	28	5	0.06
miles500	19	20	190	0.9	20	190	5	16.47
miles750	30	31	465	2.1	31	465	–	191.52
queen5_5	4	5	10	0.5	5	10	5	0.02
queen7_7	6	7	21	0.5	7	21	3	0.44
queen8_12	11	12	66	1.9	12	66	2	3.59
queen10_10	11	10	45	3.8	12	45	1	1.03
queen11_11	10	11	55	2.5	11	55	2	31.86
queen12_12	11	12	66	3.5	21	113	–	75.6
queen13_13	12	13	78	2.6	16	89	–	216.17
queen14_14	13	14	91	6.8	26	149	–	39.46
queen15_15	14	15	105	6.4	15	105	2	281.36
queen16_16	15	16	120	6.7	27	188	–	27.92
fpsol2.i.1	64	65	2080	209.9	65	2080	2	640.15
fpsol2.i.2	29	30	435	20	30	435	1	1554.87
fpsol2.i.3	29	30	435	52.5	31	457	–	1618.02
inithx.i.1	53	54	1431	897.7	54	1431	3	554.71
inithx.i.2	10	11	55	0.3	11	55	1	3.15
inithx.i.3	30	31	465	14.2	45	721	–	147.88
mulsol.i.1	48	49	1176	4.4	50	1224	–	42.94
mulsol.i.2	30	31	465	6.4	35	578	–	123.85
mulsol.i.3	30	31	465	6.5	34	551	–	24.99
mulsol.i.4	30	31	465	6.8	31	465	1	38.21
mulsol.i.5	30	31	465	6.9	36	545	–	39.05
zeroin.i.1	48	49	1176	12.3	49	1176	3	72.91
zeroin.i.2	29	30	435	11.4	34	540	–	36.05
zeroin.i.3	29	30	435	5.6	31	462	–	26.53

Table 4.4: DIMACS results comparison between *Insertion+prefiltering* [78] and *LCV*

in one critical subgraph efficiently. After the identification of such a constraint, *Insertion* removes all violations concerning other critical subgraphs. Thus the other critical subgraphs are destroyed and the search will exclusively focus on the current located critical subgraph. Second, *TabuCol*'s neighborhood structure consists in changing the color on the violated nodes exclusively. By removing all non-selected violated constraints on each iteration of *Insertion*, the

number of neighborhood solutions to evaluate in *TabuCol* is dramatically reduced, then the time consumption for *TabuCol* is very low.

The problem structure also has an important impact on the performance of *LCV*. In contrast to FAP instances, in graph k-coloring the edge constraint and color domain are uniform for all nodes in the problems. The *constructor* in *LCV* works in a *Dsatur* like constructive manner which may build a large and k-colorable subgraph. This is why even with the *centralization* procedure which attempts to reduce the subgraph size, *LCV* is much less effective than *Insertion*.

4.6 Conclusion

In this chapter, we describe the basic definitions and solution techniques for the graph k-coloring problem. Following the description of the problem, the popular benchmark DIMACS is presented with different categories of instances. Current solution methods for k-coloring are presented in two classes: heuristic and exact methods.

In the heuristic category, a roadmap of algorithm design in recent decades is introduced from *DSatur* [46] in 1979 to *TabuCol* [97] in 1987, *HCA* [106] in 1999 and *EvoDiv* [133] in 2010. This history shows that the algorithm development from simple heuristic to stochastic local search algorithms and nature inspired evolutionary approach improve the state-of-the-art for this problem.

Under the category of exact algorithms, due to the NP-hardness, the methods in the literature are still not very efficient in dealing with larger instances. From our knowledge, the state of the art of these algorithms to determine the k-colorability is proposed by Herrmann and Hertz. Their hybrid approach [138] achieves an impressive performance improvement on relatively larger instances comparing to the previous proposals.

After the description of the state of the art methods in solving the k-coloring problem in the literature, two novel data structures dedicated to accelerate the computational time of *TabuCol* for k-coloring and FAP problems are proposed. Both structures adopt a so called *Tabling Strategy* which reduces the time complexity of the evaluation step in *TabuCol* by remembering the current violation states. Our results show a significant speed gain using such an improvement. The first structure, named gamma table, stores the inconsistency state for each domain value of each variable. The second structure, named bounds list, is dedicated to the FAP; it allows *TabuCol* to accelerate the choice of the new resources during a move.

The adoption of *LCV* on the critical subgraph identification in the k-coloring problem is evaluated by comparing it to the performance of *Insertion+prefiltering* proposed by Desrosiers et al. The experimental results indicate no performance improvement of the *LCV* approach on DIMACS instances. *LCV* suffers from a non-tailored algorithm

design which is not dedicated to the k-coloring features. We conclude that the *LCV* is not suitable for a uniform structure problem such as the k-coloring problem where the constraints and the nodes domains are too uniform for the *LCV* process.

Conclusions and perspectives

In this dissertation, we have studied the problem of identifying on Irreducible Infeasible Subset in the context of a constraint satisfaction problem, and more specifically, a binary constraint satisfaction problem. The thesis is organized as follows: the first two chapters introduce the state-of-the-art, definitions and solution techniques, of the Constraint Satisfaction Problem (CSP) and Irreducible Infeasible Subset (IIS). The third chapter introduces our contribution to IIS identification with a new method called *LCV* for *Locator-Constructor-Verificator*, and its application to the Frequency Assignment Problem (FAP). In the fourth chapter, we propose some enhancement on *TabuCol* data structure to speedup the algorithm and we apply *LCV* method to identify critical subgraphs in the graph k-coloring problem. These chapters are summarized as follows.

In the first chapter, we have introduced the definition of the constraint satisfaction problem (CSP) which is represented by a triple of a finite variables set, a finite constraints set and a finite domains set. Three local consistency levels, namely Node-Consistency, Arc-Consistency and Path-Consistency, are introduced to identify the consistency on a subset of variables and constraints of the problem. We have detailed several algorithms for Arc-Consistency, and their time/space complexity are also compared. From the comparison, we have selected the *AC3* algorithm to be embedded in our method for IIS search thanks to its simplicity and effectiveness in domain filtering.

In addition to the Arc-Consistency algorithms, the *deadend learning* is also an effective technique to reduce the search space. The definitions of *deadend* and *nogood* are given in the chapter to ease the description of learning methods. The *deadend learning* technique generates supplementary constraints to cut the branches of the search tree under which it is not possible to find feasible solutions. Despite of their knowledge generating capability, the learning techniques still need to improve their time and space complexity to be used efficiently.

The solution techniques of the CSP are presented under exact and heuristic categories. Under the exact approach, several backtracking algorithms are introduced as main players to solve the CSP. These methods can provide an exact satisfiability/unsatisfiability proof of the problem. Among them, the MAC algorithm (Maintaining Arc-Consistency

during search) shows its simplicity and effectiveness by embedding domain filtering propagation inside the backtracking. It is chosen to provide the unsatisfiability proof in our method for IIS identification. Alongside the backtracking algorithms, numerous heuristics have been also illustrated; they provide a compromise between computational time and solution quality on large size instances.

The second chapter presents on the main topic of this dissertation - Infeasible Subset (IS) and Irreducible Infeasible Subset (IIS). The definitions of IS and IIS are given at the beginning of the chapter and followed by important properties of IS and IIS. We restate in this conclusion that an IS is a CSP subproblem for which there is no solution and an IIS is an IS which becomes feasible by removing any of its constraints or variables. We have also defined the notion of a critical variable and constraint. The solution techniques in dealing with IIS in the literature are described under two approaches: satisfiability testing and hitting set approach.

The satisfiability testing approach consists of a satisfiability solver and adding/removing heuristic techniques. The hitting set approach consists in iteratively generating the violated constraints subsets by a MaxCSP solver and finding a hitting set on the basis of these violated constraints subsets. These two approaches share common iterative execution mechanisms and are distinguished by using a different embedded core solver. Our theoretical analysis concludes that the hitting set approach is more suitable for applications for which the satisfiability is difficult to prove, and the maximal satisfaction solutions are conveniently obtained. At the end of the chapter, we have detailed the *Insertion* algorithm, which is proposed by Galinier and Hertz [3], and extended by Desrosiers et al. [78] to deal with the frequency planning problem. We have shown the drawback of the *Insertion* algorithm on the FAP and explained the motivation of a new method to deal with FAP instances.

The third chapter launches the study of IIS identification on our target operational research problem, the frequency planning. The chapter starts with a brief introduction of the FAP and its CSP modeling. After clarifying the connection between the FAP and CSP, the benchmarks used to evaluate our approach are illustrated by their characteristics and topologies. We have used four benchmarks with several instances: CELAR (11 instances), GRAPH (9 instances), ROADEF2001 (40 instances with different levels of constraint relaxation i.e. 220 instances at all) and SOES (20 instances). For CELAR and GRAPH we have converted the instances into infeasible ones by removing some upper constraints from the variables domain. For ROADEF2001 we have kept the infeasible levels of relaxation, and for SOES2008 the n-ary constraints are not used. All these problems are RLFAP based that is the Radio Link Frequency Assignment Problem but ROADEF2001 and SOES2008 are frequency and polarization planning problems while CELAR and GRAPH do not include polarization planning. IIS results were already published on CELAR and GRAPH; IIS results were given by the CELAR on SOES2008 instances; and there were no previous results for ROADEF2001 when we did the work.

An initial algorithm, called the *2-phase* algorithm, is conceived to address some important design aspects on searching for an IIS on the FAP instances. The algorithm is based on a construction phase (the selection of a constraint to detect the IIS) and a verification phase (verification of the infeasibility/feasibility of the set of selected constraints by the exact algorithm MAC). Something quite important there is that the algorithm tries to build the IIS by exploiting the connectivity property of the IIS. When the first constraint of one IIS is identified, its adjacent constraints are considered during IIS construction. Indeed, the IIS is a connected subgraph, so if the construction of one IIS respects the connectivity among the constraints, the construction is more efficient by concentrating the search around some initial constraints. In this procedure we have applied a technique of saturation which consists in completing the constraints among the critical variables. Using the saturation technique, the number of executions of *MinConflict*() is reduced substantially. In comparison with Hemery et al. [61] on CELAR instances, the results were promising but the algorithm was not efficient in the construction phase.

Based on the experimental analysis on the *2-phase* algorithm, our new proposal of the *LCV* algorithm is described for IS search. The *LCV* consists of three independent components which are named *locator*, *constructor* and *verificator*. The routines of *LCV* can be described briefly as follows: *locator* scans the entire problem and attempts to locate a critical constraint potentially inside an IS; *constructor* constructs a potential IS by adding one by one the adjacent constraints around the constraint identified by *locator*; and finally, *verificator* provides the proof of feasibility or infeasibility of the subproblem built by the *constructor*. The *verificator* routine still uses a MAC algorithm to check the infeasibility of the constraint subset. *Contructor* uses several heuristics such as the saturation technique to introduce new constraints, minimum conflict technique, and minimum remaining value to find a frequency assignment which minimizes constraint violation. *Locator* uses the *AC3* algorithm as a pre-filtering step, we have seen that it accelerates the IIS search speed, and an embedded local search in the *BreakScan* routine to minimize the violations on highly weighted constraints. A comparison between *Hill Climbing* and *TabuCol* is done as the local search procedure in *BreakScan* and has emphasized the improved performance of *Hill Climbing*.

Then in order to iteratively identify a smaller IS inside one IS until no smaller IS can be found, two detector routines for IIS of variables and IIS of constraints have also been introduced to extract the IIS on the basis of the IS found by *LCV*. The IIS of variables detector firstly locates a small critical constraint subset H inside the entering IS. Secondly, the detector iteratively adds one variable into the located core formed by the critical constraints H and its variables. The variable selection is based on the MRV heuristic. The procedure stops when the MAC algorithm proves that the core is unsatisfiable. The other detector is similar but it adds constraints to the core and does not apply a saturation technique.

The new *LCV+detector* approach has shown a significant performance improvement when dealing with the CELAR and GRAPH instances. From the comparison with Hemery et al. [61], the proposed *LCV+detector* algorithm surpasses the performance of wcore on both IIS size and computational time. Also we have observed that *LCV+detector of constraints* offers the best performance. These results were confirmed on ROADEF2001 on which we compared the sizes of IS (obtained with *LCV*), IIS of variables and IIS of constraints (obtained with *LCV* plus the respective detector). Finally concerning SOES2008 instances, we have compared our *LCV* results with the results given by the *SSA* algorithm from DGA/CELAR and the performance of *LCV* was better for all cases.

In the fourth chapter, considering the existing results on FAP application, we apply the *LCV* algorithm on critical subgraph identification to the graph k-coloring problem. The chapter of critical subgraph identification begins with the definition of the k-coloring problem and the introduction of some exact and heuristic solution techniques in the literature. We have observed that *TabuCol* is still a competitive candidate to provide an excellent compromise between solution quality and computational time. By analysis the behavior of *TabuCol*, we have proposed two novel data structures to accelerate performance on both FAP and k-coloring instances. These structures are called *gamma table* and *bounds list* table. The *gamma table* stores the number of inconsistency values for each value of the variable domain in a decentralized way. The second structure is dedicated to the FAP problem; it introduces bounds describing the variable domain. The bounds come issued from the T-coloring constraints and they allow *TabuCol* to speedup the inconsistency checking of each value by considering the value outside the bounds and inside the bounds of the interval of the frequency spectrum. We evaluate formally the time and space complexity of the move evaluation and move update of *TabuCol* with the proposed data structures.

At the end of the chapter, we have compared the results obtained by *LCV* and Desrosiers et al. *Insertion+prefiltering* algorithm [78]. The results of *LCV* on several DIMACS instances were not impressive on runtime performance and size of critical subgraph. We have underlined the problem for *LCV* with the uniform representation of constraints and domains in the k-coloring problem, and further, the heuristic used in *constructor* to extend the subproblem. The design of *LCV* cannot take advantage on the structure of the k-coloring problem, while *Insertion* can benefit from the combination of its constraints pruning mechanism and embedded *TabuCol* in dealing with the same problem. We have concluded that, currently, our proposed *LCV* is not efficient and effective for the critical subgraph identification in the graph k-coloring problem.

After our current research on IIS detection on frequency planning and graph k-coloring, there is still the possibility of exploring new solutions. As mentioned before, the results of *LCV* on the frequency planning application were very impressive while the results on graph k-coloring cannot achieve the same performance as the state of the art method *Insertion+prefiltering*. The preliminary analysis casts a doubt on the heuristic used in the *constructor* routine of *LCV*

for k-coloring. Thus, one possible research avenue will focus on the proposal of new heuristics to construct the critical constraints subset effectively. Alongside *constructor*, *locator* also can be modified to improve IS detection. Instead of locating one constraint, several constraints can be considered at same time.

Further, the research may concentrate on identifying all the IIS of a problem. All IIS detection may bring two benefits. Firstly, since one IIS indicates the unsatisfiability of the problem and the constraint violation in a small part of the problem, even with the relaxation of constraints in such an IIS, it still may not change the global unsatisfiability of a problem in case that there exists several IIS. With the identification of all IIS and constraint relaxation, we may propose a complete solution to modify the problem modeling. Secondly, all IIS detection may provide a compromise MaxCSP solution. Regarding the relation between the IIS detection and MaxCSP solution, one MaxCSP solution can be achieved by violating the constraints which form a minimum hitting set on the collection of all IIS.

The third research direction can be focused on the IIS detection in the problem with high-arity constraints. Such objective attracts more attention recently in frequency planning application due to the fact that modern modeling techniques of FAP involve high-arity constraints as described in Section 3.2.3. Such a challenge explores a different domain and may bring the benefits on IIS detection with intermodulation and perturbation constraints. If this problem is solved, the conversion between n-ary constraints and binary constraints will be unnecessary in IIS detection for n-ary problems.

References

1. Chinneck, J.W.: Feasibility and Infeasibility in Optimization: Algorithms and Computational Methods. 1st edn. Springer Publishing Company, Incorporated (2007)
2. van Loon, J.N.M.: Irreducibly inconsistent systems of linear inequalities. European Journal of Operational Research **8**(3) (November 1981) 283–288
3. Galinier, P., Hertz, A.: Solution techniques for the large set covering problem. Discrete Applied Mathematics **155**(3) (2007) 312–326
4. Freuder, E., Wallace, R.: Partial constraint satisfaction. Artifical Intelligence **58**(1–3) (1992) 21–70
5. Mackworth, A.: Consistency in networks of relations. Artificial Intelligence **8(1)** (1977) 99–118
6. Montanari, U.: Networks of constraints: Fundamental properties and applications to picture processing. Information Science **7** (1974) 95–132
7. Freuder, E.: A sufficient condition for backtrack-free search. Journal of the ACM **29(1)** (Jan 1982) 24–32
8. C.Bessière: Constraint propagation. Technical report, LIRMM 06020, CNRS/University of Montpellier (March 2006)
9. Zhang, Y., Yap, R.: Making AC-3 an optimal algorithm. In: Proceedings IJCAI'01 Seattle WA. (2001) 316–321
10. Mohr, R., Henderson, T.: Arc and path consistency revised. Artificial Intelligence **28** (1986) 225–233
11. Bessière, C.: Arc-consistency and arc-consistency again. Artificial Intelligence **65** (1994) 179–190
12. C.Bessière, Régin, J.: Refining the basic constraint propagation algorithm. In: Proceedings IJCAI'01 Seattle WA. (2001) 309–315
13. Mehta, D., van Dongen, M.: Two new lightweight arc consistency algorithms. In: Proceedings of the CP'04 Workshop on Constraint Propagation and Implementation, Toronto, Canada. (2004) 109–123
14. Boussemart, F., Hemery, F., Lecoutre, C.: Revision ordering heuristics for the constraint satisfaction problem. In: Proceedings of the CP'04 Workshop on Constraint Propagation and Implementation, Toronto, Canada. (2004) 29–44
15. Mackworth, A.K.: On reading sketch maps. In: Proceedings of the 5th international joint conference on Artificial intelligence - Volume 2, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc. (1977) 598–606
16. Wallace, R.: Why AC-3 is almost always better than AC-4 for establishing arc consistency in csps. In: Proceedings IJCAI'93 Chambéry, France. (1993) 239–245
17. Berlandier, T.: Improving domain filtering using restricted path consistency. In: Proceedings IEEE Conference on Artificial Intelligence and Applications CAIA '95. (1995)
18. Freuder, E., Elfe, C.: Neighborhood inverse consistency preprocessing. In: Proceedings AAAI'96, Portland OR, USA (1996) 202–208
19. Debruyne, R., Bessière, C.: From restricted path consistency to max-restricted path consistency. In: Proceedings IJCAI'97, Linz, Austria (1997) 312–326

20. Debruyne, R.: A property of path inverse consistency leading to an optimal PIC algorithm. In: Proceedings ECAI'00, Berlin, Germany (2000) 88–92
21. Dechter, R., Pearl, J.: Network-based heuristics for constraint satisfaction problems. Artificial Intelligence **34** (1988) 1–38
22. Haralick, M., Elliot, G.: Increasing tree-search efficiency for constraint satisfaction problems. Artificial Intelligence **14** (1980) 263–313
23. Frost, D., Dechter, R.: Dead-end driven learning. In: Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94). (1994) 294–300
24. Gaschnig, J.: Performance measurement and analysis of search algorithms. Technical report, CMU-CS-79-124, Carnegie Mellon University (1979)
25. Dechter, R.: Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition. Artificial Intellignece **41** (1990) 273–312
26. Katsirelos, G., Bacchus, F.: Unrestricted nogood recording in csp search. In: IN PRINCIPLES AND PRACTICE OF CONSTRAINT PROGRAMMING, Springer (2003) 873–877
27. Bayardo, J.R.J., Miranker, D.P.: A complexity analysis of space-bounded learning algorithms for the constraint satisfaction problem. In: Proceedings of the thirteenth national conference on Artificial intelligence - Volume 1. AAAI'96, AAAI Press (1996) 298–304
28. Dib, M., Caminada, A., Mabed, H.: Nogood recording with tabu search for csp (application to fap). In: Asia International Conference on Modelling and Simulation. (2009) 218–223
29. Sabin, D., Freuder, E.: Contradicting conventional wisdom in constraint satisfaction. In: Proceedings of ECAI'94. (1994) 125–129
30. Bessière, C., Régin, J.: MAC and combined heuristics : two reasons to forsake FC (and CBJ?) on hard problems. In: Proceedings of CP'96. (1996) 61–75
31. van Dongen, M.: AC3_d an efficient arc-consistency algorithm with a low space-complexity. Technical report, TR-01-2002, Cork Constraint Computation Centre, CS Department, University College Cork, Cork, Ireland (2002)
32. van Dongen, M.: Lightweight MAC algorithms. Technical report, TR-02-2003, Cork Constraint Computation Centre, CS Department, University College Cork, Cork, Ireland (April 2003)
33. Likitvivatanavong, C., Zhang, Y., Bowen, J., Freuder, E.: Arc consistency in MAC: A new perspective. In van Dongen, M., ed.: Proceedings Constraint Propagation and Implementation, first International workshop Toronto, Canada. (September 2004) 93–108
34. Régin, J.: Maintaining arc consistency algorithms during the search with an optimal time and space complexity. In van Dongen, M., ed.: Proceedings Constraint Propagation and Implementation, first International workshop Toronto, Canada. (September 2004) 125–138
35. Ginsberg, M.: Dynamic backtracking. Journal of Artificial Intelligence Research **1** (1993) 25–46
36. Jussien, N., Debruyne, R., Boizumault, P.: Maintaining arc-consistency within dynamic backtracking. In: Proceedings of Principles and Practice of Constraint Programming. (2000) 249–261
37. Karp, R.M.: Reducibility among combinatorial problems. In Jünger, M., Liebling, T.M., Naddef, D., Nemhauser, G.L., Pulleyblank, W.R., Reinelt, G., Rinaldi, G., Wolsey, L.A., eds.: 50 Years of Integer Programming 1958-2008. Springer Berlin Heidelberg (2010) 219–241
38. Russell, S.J., Norvig, P., Candy, J.F., Malik, J.M., Edwards, D.D.: Artificial intelligence: a modern approach. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1996)

39. Beck, J.C., Prosser, P., Wallace, R.J.: Toward understanding variable ordering heuristics for constraint satisfaction problems. In: In: Proceedings of the Fourteenth Irish Artificial Intelligence and Cognitive Science Conference. (2003) 11–16
40. Wallace, R.J.: Determining the principles underlying performance variation in csp heuristics. (2008) 857–880
41. Vernooy, M., Havens, W.S.: An examination of probabilistic value-ordering heuristics. In: Proceedings of the 12th Australian Joint Conference on Artificial Intelligence: Advanced Topics in Artificial Intelligence. AI '99, London, UK, Springer-Verlag (1999) 340–352
42. Frost, D., Dechter, R.: Look-ahead value ordering for constraint satisfaction problems. In: Proceedings of International Joint Conference on Artificial Intelligence IJCAI-95. (1995) 572–578
43. Lecoutre, C., Sais, L., Tabary, S., Vidal, V.: Nogood recording from restarts. In: Proceedings of the 20th international joint conference on Artifical intelligence. IJCAI'07, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc. (2007) 131–136
44. Wallace, R.: Analysis of heuristic synergies. In Hnich, B., Carlsson, M., Fages, F., Rossi, F., eds.: Recent Advances in Constraints. Volume 3978 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2006) 73–87
45. Minton, S., Johnston, M.D., Philips, A.B., Laird, P.: Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems. Artificial Intelligence **58**(1-3) (1992) 161–205
46. Brélaz, D.: New methods to color the vertices of a graph. Commun. ACM **22**(4) (1979) 251–256
47. Morris, P.: The breakout method for escaping from local minima. In: Proceeceding of the 11th National Conference on Artificial Intelligence, Washington, DC (1993) 40–45
48. Dupont, A., Alvernhe, E., Vasquez, M.: Efficient filtering and tabu search on a consistent neighbourhood for the frequency assignment problem with polarisation. Annals of Operations Research **130** (2004) 179–198 10.1023/B:ANOR.0000032575.38969.ab.
49. Dib, M., Caminada, A., Mabed, H.: Nogood recording with tabu search for csp (application to fap). In: Asia International Conference on Modelling and Simulation. (2009) 218–223
50. Galinier, P., Hao, J.K.: A general approach for constraint solving by local search. J. Math. Model. Algorithms **3**(1) (2004) 73–88
51. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Commun. ACM **5**(7) (1962) 394–397
52. Gu, J., Purdom, P.W., Franco, J., Wah, B.W.: Algorithms for the satisfiability (SAT) Problem: A survey. In Du, D.Z., Gu, J., Pardalos, P., eds.: Satisfiability Problem: Theory and applications. DIMACS Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society (1997) 19–152
53. Hirsch, E.A.: SAT local search algorithms: Worst-case study. Journal of Automated Reasonning **24**(1/2) (February 2000) 127–143
54. Hoos, H.H., Stützle, T.: Local search algortihms for SAT: An empirical evaluation. Journal of Automated Reasoning **24**(4) (2000) 421–481
55. Lynce, I., Baptista, L., Marques-Silva, J.: Stochastic systematic search algorithms for satisfiability. Electronic Notes in Discrete Mathematics **9**(0) (2001) 190–204
56. van Loon, J.N.M.: Irreducibly inconsistent systems of linear inequalities. European Journal of Operational Research **8**(3) (November 1981) 283–288
57. Chinneck, J.W.: Finding a useful subset of constraints for analysis in an infeasible linear program. Informs Journal on Computing **9**(2) (1997) 164–174
58. Gleeson, J., Ryan, J.: Identifying minimally infeasible subsystems of inequalities. ORSA Journal on Computing **2**(1) (1990) 61–63
59. Greenberg, H.J., Murphy, F.H.: Approaches to diagnosing infeasible linear programs. INFORMS Journal on Computing **3**(3) (1991) 253–261

60. Grégoire, É., Mazure, B., Piette, C.: Local-search extraction of MUSes. Constraints **12**(3) (2007) 325–344
61. Hemery, F., Lecoutre, C., Sais, L., Boussemart, F.: Extracting MUCs from constraint networks. In: ECAI. (2006) 113–117
62. Marques-Silva, J.: Minimal unsatisfiability: Models, algorithms and applications (invited paper). In: Multiple-Valued Logic (ISMVL), 2010 40th IEEE International Symposium on. (May 2010) 9–14
63. Grégoire, É., Mazure, B., Piette, C.: On approaches to explaining infeasibility of sets of boolean clauses. In: Proceedings of the 2008 20th IEEE International Conference on Tools with Artificial Intelligence - Volume 01, Washington, DC, USA, IEEE Computer Society (2008) 74–83
64. Piette, C., Hamadi, Y., Saïs, L.: Efficient combination of decision procedures for mus computation. In: Proceedings of the 7th international conference on Frontiers of combining systems. FroCoS'09, Berlin, Heidelberg, Springer-Verlag (2009) 335–349
65. Boussemart, F., Hemery, F., Lecoutre, C., Sais, L.: Boosting systematic search by weighting constraints. In: Proceedings of ECAI'04. (2004) 146–150
66. Grégoire, É., Mazure, B., Piette, C.: On finding minimally unsatisfiable cores of CSPs. International Journal on Artificial Intelligence Tools **17**(4) (2008) 745–763
67. Zhang, L., Malik, S.: Extracting small unsatisfiable cores from unsatisfiable boolean formula. In: Proceedings of Theory and Applications of satisfiability Testing (SAT'03). (2003)
68. Zhang, L.: Searching for truth: techniques for satisfiability of boolean formulas. PhD thesis, Princeton, NJ, USA (2003) AAI3102236.
69. Oh, Y., Mneimneh, M.N., Andraus, Z.S., Sakallah, K.A., Markov, I.L.: AMUSE: a minimally-unsatisfiable subformula extractor. In: DAC. (2004) 518–523
70. Eisenberg, C., Faltings, B.: Using the Breakout Algorithm to Identify Hard and Unsolvable Subproblems. In: Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming (CP-2003), Lecture Notes in Computer Science. (2003)
71. Mazure, B., Saïs, L., Grégoire, É.: Boosting complete techniques thanks to local search methods. Annals of Mathematics and Artificial Intelligence **22** (January 1998) 319–331
72. Kautz, H., Selman, B., McAllester, D.: Walksat in the sat 2004 competition. In: International Conference on Theory and Applications of Satisfiability Testing (SAT'04). (2004)
73. Van Maaren, H., Wieringa, S.: Finding guaranteed muses fast. In: Proceedings of the 11th international conference on Theory and applications of satisfiability testing. SAT'08, Berlin, Heidelberg, Springer-Verlag (2008) 291–304
74. de la Banda, M.G., Stuckey, P.J., Wazny, J.: Finding all minimal unsatisfiable subsets. In: PPDP '03: Proceedings of the 5th ACM SIGPLAN international conference on Principles and practice of declaritive programming, New York, NY, USA, ACM (2003) 32–43
75. Bruni, R., Sassano, A.: Finding minimal unsatisfiable subformulae in satisfiability instances. In: CP. (2000) 495–499
76. Liffiton, M.H., Sakallah, K.A.: On finding all minimally unsatisfiable subformulas. In: in International Conf. on Theory and Applications of Satisfiability Testing, Springer-Verlag (2005) 173–186
77. Bruni, R.: Approximating minimal unsatisfiable subformulae by means of adaptive core search. Discrete Applied Mathematics **130**(2) (2003) 85–100
78. Desrosiers, C., Galinier, P., Hertz, A.: Efficient algorithms for finding critical subgraphs. Discrete Appl. Math. **156** (2008) 244–266
79. Bailey, J., Manoukian, T., Ramamohanarao, K.: A fast algorithm for computing hypergraph transversals and its application in mining emerging patterns. In: Proceedings of the Third IEEE International Conference on Data Mining. ICDM '03, Washington, DC, USA, IEEE Computer Society (2003) 485

80. Bailey, J., Stuckey, P.J.: Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization. In: In Proc. of the 7th International Symposium on Practical Aspects of Declarative Languages (PADL05, Springer (2005) 174–186
81. Liffiton, M.H., Moffitt, M.D., Pollack, M.E., Sakallah, K.A.: Identifying conflicts in overconstrained temporal problems. In: Proceedings of the 19th international joint conference on Artificial intelligence, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc. (2005) 205–211
82. Desrosiers, C., Galinier, P., Hertz, A., Paroz, S.: Using heuristics to find minimal unsatisfiable subformulas in satisfiability problems. Journal of Combinatorial Optimization **18**(2) (2009) 124–150
83. Liffiton, M., Sakallah, K.: Algorithms for computing minimal unsatisfiable subsets of constraints. Journal of Automated Reasoning **40** (2008) 1–33
84. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: engineering an efficient sat solver. In: Proceedings of the 38th annual Design Automation Conference. DAC '01, New York, NY, USA, ACM (2001) 530–535
85. Fu, Z., Malik, S.: On solving the partial MAX-SAT problem. In: SAT. (2006) 252–265
86. Aardal, K., van Hoesel, S., Koster, A., Mannino, C., Sassano, A.: Models and solution techniques for frequency assignment problems. In: (Updated version of a paper appeared in 4OR 1, 261-317, 2003). (Jan 2007)
87. Galinier, P., Hertz, A.: A survey of local search methods for graph coloring. Comput. Oper. Res. **33**(9) (2006) 2547–2562
88. Gondran, A.: Modélisation et optimisation des réseaux locaux sans fil Wireless local area network modeling and optimization. PhD thesis, Universite de Technologie de Belfort Montbeliard (2008)
89. Mabed, H., Caminada, A., Hao, J.K., Renaud, D.: A dynamic traffic model for frequency assignment. In: PPSN. (2002) 779–788
90. Dib, M.: Tabu-NG:hybridation de programmation par contraintes et recherche locale pour la résolution de CSP. PhD thesis, Universite de Technologie de Belfort Montbeliard (2010)
91. Gondran, A., Baala, O., Mabed, H., Caminada, A.: Hypergraph t-coloring for automatic frequency planning problem in wireless lan. In: PIMRC. (2008) 1–5
92. Cabon, B., de Givry, S., Lobjois, L., Schiex, T., Warners, J.P.: Radio link frequency assignment. Constraints **4**(1) (1999) 79–89
93. van Benthem, H.: Graph: generating radiolink frequency assignment problems heuristically. (1995)
94. Koster, A.M.C.A., van Hoesel, S.P.M., Kolen, A.W.J.: The partial constraint satisfaction problem: Facets and lifting theorems. Oper. Res. Lett. **23**(3-5) (1998) 89–97
95. Defaix, T.: FAPP – problèmes d'allocation de fréquences avec polarisation. `http://www.fap.ema.fr/Local/fap/dir/documents/fapp_roadef-fr-rev3.pdf` (november 2003)
96. Hu, J., Galinier, P., Caminada, A.: On identifying infeasible subsets in constraint satisfaction problems. In: ICAI. (2010) 615–619
97. Hertz, A., Werra, D.: Using tabu search techniques for graph coloring. Computing **39** (1987) 345–351
98. Galinier, P., Gendreau, M., Soriano, P., Bisaillon, S.: Solving the frequency assignment problem with polarization by local search and tabu. 4OR: A Quarterly Journal of Operations Research **3**(1) (2005) 59–78
99. Hu, J., Galinier, P., Caminada, A.: Yet another breakout inspired infeasible subset detection in constraint satisfaction problem. In: ICAI. (2011)
100. Mehrotra, A., Trick, M.: A column generation approach for graph coloring. INFORMS Journal On Computing **8**(4) (1996) 344–354
101. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman (1979)
102. Diestel, R.: Graph Theory (Graduate Texts in Mathematics). Springer (August 2005)

103. In Johnson, D.S., Trick, M.A., eds.: Cliques, coloring, and satisfiability: 2nd DIMACS implementation challenge 1993. DIMACS series in discrete mathematics and theoretical computer science. American Mathematical Society (1996)
104. Chiarandini, M., Dumitrescu, I., Stützle, T.: Stochastic local search algorithms for the graph colouring problem. In Gonzalez, T.F., ed.: Handbook of Approximation Algorithms and Metaheuristics. Computer & Information Science Series. Chapman & Hall/CRC, Boca Raton, FL, USA (2007) 63.1–63.17
105. Malaguti, E., Toth, P.: A survey on vertex coloring problems. International Transactions in Operational Research **17**(1) (2010) 1–34
106. Galinier, P., Hao, J.K.: Hybrid evolutionary algorithms for graph coloring. J. Comb. Optim. **3**(4) (1999) 379–397
107. Kumar, V.: Algorithms for constraint satisfaction problems: A survey. AI MAGAZINE **13**(1) (1992) 32–44
108. Lu, Z., Hao, J.K.: A memetic algorithm for graph coloring. European Journal of Operational Research **203**(1) (2010) 241–250
109. Leighton, F.T.: A graph coloring algorithm for large scheduling problems. Journal of Research of the National Bureau of Standards **84**(6) (1979) 489–506
110. Johnson, D.S., Aragon, C.R., McGeoch, L.A., Schevon, C.: Optimization by simulated annealing: An experimental evaluation part II, graph coloring and number partitioning. Operations Research **39**(3) (1991) 378–406
111. Bollobás, B., Thomason, A.: Random graphs of small order. Annals of Discrete Mathematics **28** (1985) 251–256
112. Hoos, H., Sttzle, T.: Stochastic Local Search: Foundations & Applications. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2004)
113. Chams, M., Hertz, A., de Werra, D.: Some experiments with simulated annealing for coloring graphs. European Journal of Operational Research **32**(2) (1987) 260–266 Third EURO Summer Institute Special Issue Decision Making in an Uncertain World.
114. Johnson, D.S., Aragon, C.R., McGeoch, L.A., Schevon, C.: Optimization by simulated annealing: an experimental evaluation; part ii, graph coloring and number partitioning. Oper. Res. **39** (May 1991) 378–406
115. Blöchliger, I., Zufferey, N.: A reactive tabu search usin gpartial solutions for the graph coloring problem. Computers and Operations Research **35**(3) (2008) 960–975
116. Avanthay, C., Hertz, A., Zufferey, N.: A variable neighborhood search for graph coloring. European Journal of Operational Research **151** (2003) 379–388
117. Mladenovic, N., Hansen, P.: Variable neighborhood search. Computers Operations Research **24** (1997) 1097–1100
118. Hertz, A., Plumettaz, M., Zufferey, N.: Variable space search for graph coloring. Discrete Applied Mathematics (156) (2002) 2551–2560
119. Mladenović, N., Plastria, F., Urošević, D.: Formulation space search for circle packing problems. In: Proceedings of the 2007 international conference on Engineering stochastic local search algorithms: designing, implementing and analyzing effective heuristics. SLS'07, Berlin, Heidelberg, Springer-Verlag (2007) 212–216
120. Denzinger, J., Fuchs, M., Fuchs, M., Informatik, F.F., Munchen, T.: High performance ATP systems by combining several AI methods. In: In Proc. Fifteenth International Joint Conference on Artificial Intelligence (IJCAI 97, Morgan Kaufmann (1997) 102–107
121. Chiarandini, M., Stuetzle, T.: An application of iterated local search to graph coloring. In: Proceedings of the Computational Symposium on Graph Coloring and its Generalizations, New York, USA (2002) 112–125
122. Devarenne, I., Caminada, A., Mabed, H.: Analysis of adaptive local search for graph coloring problem. In: MIC2005. (2005) (1204)1–(1204)6
123. DIB, M., ABDALLAH, R., CAMINADA, A.: Tabu–NG : Une approche de résolution hybride pour la coloration de graphes. In: ROADEF2011. (2011)

124. Morgenstern, C.: Distributed coloration neighborhood search. In: Second DIMACS Implementation Challenge, DIMACS series in Discrete Mathematics and Theoretical Computer Science. (1996) 335–357
125. Holland, J.H.: Adaptation in natural and artificial systems. MIT Press, Cambridge, MA, USA (1992)
126. Ashlock, D.: Evolutionary Computation for Modeling and Optimization. Springer (2004)
127. Davis, L.: Order-based genetic algorithms and the graph coloring problem. In: Handbook of Genetic Algorithms. Van Nostrand Reinhold; New York (1991) 72–90
128. Glass, C.A., Prügel-Bennett, A.: Genetic algorithm for graph colouring: Exploration of galinier and hao's algorithm. Journal of Combinatorial Optimization **7** (2003) 229–236
129. Galinier, P., Hertz, A., Zufferey, N.: An adaptive memory algorithm for the k-coloring problem. Discrete Applied Mathematics **156**(2) (2008) 267–279
130. Rochat, Y., Taillard, E.: Probabilistic diversification and intensification in local search for vehicle routing. Journal Heuristics (1) (1995)
131. Malaguti, E., Monaci, M., Toth, P.: A metaheuristic approach for the vertex coloring problem. INFORMS Journal on Computing **20**(2) (2008) 302–316
132. Caprara, A., Fischetti, M., Toth, P.: A heuristic method for the set covering problem. Oper. Res. **47** (May 1999) 730–743
133. Porumbel, D.C., Hao, J.K., Kuntz, P.: An evolutionary approach with diversity guarantee and well-informed grouping recombination for graph coloring. Computers & Operations Research **37** (2010) 1822–1832
134. Brown, J.R.: Chromatic scheduling and the chromatic number problem. Management Science **19**(4) (1972) 456–463
135. Sewell, E.: An improved algorithm for exact graph coloring. In: DIMACS Series in Discrete Mathematics and Theoretical Computer Science. (1996) 359–373
136. Peemöller, J.: A correction to Brelaz's modification of Brown's coloring algorithm. Communications of the ACM archive **26**(8) (August 1983) 595–597
137. Hansen, P., Labbé, M., Schindl, D.: Set covering and packing formulations of graph coloring: Algorithms and first polyhedral results. Discrete Optimization **6**(2) (2009) 135–147
138. Herrmann, F., Hertz, A.: Finding the chromatic number by means of critical graphs. ACM Journal of Experimental Algorithmics **7** (2002) 10
139. Gualandi, S., Malucelli, F.: Exact solution of graph coloring problems via constraint programming and column generation. INFORMS Journal on Computing (2011)

SPIM

■ École doctorale SPIM - Université de Technologie Belfort-Montbéliard
F - 90010 Belfort Cedex ■ **tél. +33 (0)3 84 58 31 39**
■ **ed-spim@univ-fcomte.fr** ■ www.ed-spim.univ-fcomte.fr

utbm
université de technologie
Belfort-Montbéliard