Christian Bessiere Brahim Hnich Zeynep Kiziltan
Toby Walsh (Eds.)

Modelling and Solving Problems with Constraints

Fifth International Workshop
Edinburgh, Scotland, 31 July 2005
Proceedings

Held in conjunction with the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI 2005)

Preface

Constraint Programming (CP) is a powerful technology to solve combinatorial problems which are ubiquitous in academia and industry. The last ten years have witnessed significant research devoted to modelling and solving problems with constraints. CP is now a mature field and has been successfully used for tackling a wide range of real-life complex applications.

As constraint solving is intractable in general, problems can become difficult to solve as their size increase. Therefore, there is always a need for more efficient solvers to cope with ever difficult problems. Techniques such as the design of specialised filtering algorithms for recurring constraints, sophisticated search techniques, heuristics to guide the search, symmetry breaking have significant impact on the time spent to solve problems. Efficiency can be improved also by bridging the gap between CP and the other communities such as Operations Research, Local Search, SAT, Planning, and Machine Learning.

Formulating an effective model for a given problem often requires trying alternate models and using "modelling tricks" such as redundant modelling and channelling. This could be a challenge even for modelling experts. The increasing use of CP necessitates higher level modelling languages to facilitate the exploitation of the available technology and to make CP reachable to a wider user base. The hope is that the next generation modelling languages will assist modellers by for instance helping acquire and validate constraints, automatically generating alternate models and selecting the most appropriate one for the application in hand, and synthesising propagators for complex constraints.

It is desirable to extend the classical framework for modelling and solving with constraints to adapt to some real-life scenarios. For instance, many problems contain uncertainty and thus the user may require robust solutions. In some cases, problems are over-constrained and the user has preferences for which constraints to relax. Explanations can be necessary to understand the solution process. Real-life problems are often optimisation problems and the users might want to improve the quality of their solutions as quickly as possible.

This workshop provides a forum for researchers who share these goals. It is the 5th in the series, following the successful earlier workshops held alongside ECAI 2000, IJCAI 2001, ECAI 2002, and ECAI 2004. The papers in these proceedings present research into many aspects of modelling and solving problems with constraints such as modelling, symmetry breaking, propagation algorithms, applications, hybrid systems, and extensions to the classical framework. In order to help push the field further, the workshop hosts a modelling challenge. Finally, the workshop includes an invited talk which gives insight into modelling and solving problems with constraints using an open-source constraint programming system called Choco.

We would like to thank all the authors who submitted papers; the invited speaker Narendra Jussien; the organisers of the modelling challenge; the members of the programme committee; the additional reviewers for their help, and Emmanuel Hebrard and Paolo Torroni for their support in the preparation of these proceedings.

The proceedings can be found online at `http://4c.ucc.ie/~brahim/ijcai05ws`.

June 2005

Christian Bessiere
Brahim Hnich
Zeynep Kiziltan
Toby Walsh
(Workshop Organizers)

Workshop Organizers

Christian Bessiere, LIRMM-CNRS, France
Brahim Hnich, 4C, University College Cork, Ireland
Zeynep Kiziltan, Università di Bologna, Italy (Chair)
Toby Walsh, NICTA and University of New South Wales, Australia

Programme Committee

Claire Bagley, Oracle Corporation, U.S.A.
Roman Bartak, Charles University, Czech Republic
Nicolas Beldiceanu, Ecole des Mines de Nantes, France
Carmen Gervet, IC-Parc, Imperial College London, U.K.
Esther Gelle, ABB Switzerland Corporate Research, Switzerland
Warwick Harvey, IC-Parc, Imperial College London, U.K.
Jimmy Lee, The Chinese University of Hong Kong, Hong Kong
Pedro Meseguer, IIIA-CSIC, Spain
Michela Milano, Università di Bologna, Italy
Wim Nuijten, ILOG, The Netherlands
Patrick Prosser, Glasgow University, U.K.
Jean-Francois Puget, ILOG, France
Jean-Charles Regin, ILOG, France
Thomas Schiex, Institut National de La Recherche Agronomique, France
Peter Stuckey, University of Melbourne, Australia
Edward Tsang, University of Essex, U.K.
Mark Wallace, Monash University, Australia

Modelling Challenge Organizers

Ian Gent, Ian Gent, University of St. Andrews, U.K.
Barbara Smith, 4C, University College Cork, Ireland

Additional Referees

Remi Coletta, Emmanuel Hebrard, Alessandro Zanarini

Table of Contents

The Choco Constraint Programming System

Narendra Jussien

École des Mines de Nantes, LINA FRE CNRS 2729, France

Abstract. The CHOCO (CHic, un Outil Contraintes avec des Objets – great, a constraint programming tool with objects) constraint programming system is an emanation of the French OCRE (Outil Contraintes pour la Recherche et l'Enseignement – a constraint tool for research and teaching) group. Choco is a java library for constraint satisfaction problems, constraint programming, and explanation-based constraint solving. It is built on an event-based propagation mechanism with backtrackable structures.

This talk is composed of two parts. In the first part, we will describe and illustrate the inner mechanisms and features of Choco. Then, we will focus on the modelling and solving tools of the system illustrating the different domains and constraints provided with Choco. This first part will be concluded with one of the key features of Choco: its extensibility.

But Choco is much more than a classical constraint programming system. In the second part of the talk, we will focus on the new features of our solver. Explanations as both an analysing and solving tool for constraint programming will be introduced and we will show how they are smoothly integrated within Choco, thanks to the Java programming language. Moreover, search mechanisms which were not previously integrated in a modular and generic way will be presented including the decision-repair and the logical Benders decomposition schemes.

Finally, we will conclude with the status of Choco regarding its use for teaching, research or economical purposes and call for participation in its development.

A SAT Encoding for the Social Golfer Problem

Ian P. Gent
School of Computer Science
University of St Andrews
Fife, Scotland
ipg@dcs.st-and.ac.uk

Inês Lynce
IST/INESC-ID
Technical University of Lisbon
Lisbon, Portugal
ines@sat.inesc-id.pt

Abstract

When given a combinatorial problem, one has two major tasks: to model the problem and to solve the selected model. Whilst much work in SAT algorithms is for building efficient solvers, we argue that many modeling decisions have a direct impact on the solvers performance. We focus on a particular combinatorial problem: the social golfer problem, and we show how to encode this problem into SAT. An important feature of the social golfer problem is the presence of symmetries, which can be tackled by adding more clauses to the encoding. Our empirical evaluation shows that different encodings can improve or degrade search dramatically depending on the solver. We also show empirically that by choosing the *right* encoding one may exploit the heavy-tail behavior.

1 Introduction

Recent years have seen remarkable progress in propositional satisfiability (SAT), with significant theoretical and practical contributions. Indeed, SAT solvers can currently be used to solve hard benchmark problems. State-of-the-art SAT solvers (e.g. [Moskewicz *et al.*, 2001; Goldberg and Novikov, 2002; Een and Sorensson, 2003; Kautz *et al.*, 2004; Ryan, 2004]), are with no doubt very competitive. And every year a new SAT competition is run with new solvers and new benchmarks. All solvers and benchmarks are classified according to three categories: industrial, handmade and random. Every year, almost all the previous year winners for each category are beaten by a new, more efficient solver. Also, the new solvers are able to solve part of the benchmark problems that were not solved in the previous year in a reasonable amount of time.

The progress in SAT solving has attracted the attention of researchers that usually use other technologies to solve their problems. Encoding problems in CNF format and solving them with SAT solvers is indeed a competitive approach. SAT has the advantage of being very easy in its formulation. Nonetheless, the simplicity of the CNF format makes its use very restrictive. For example, a constraint problem with a few dozen of variables may result in a SAT problem with thousands of variables and millions of clauses. Also, one may argue that a cause of inefficiency is the loss of structure during problem reductions.

Even though the SAT community is extremely motivated for continuously improving SAT solvers performance, there is much to be done with respect to SAT encodings. We believe that many applications do not benefit from the efficiency in SAT solving due to inefficiencies introduced while producing SAT encodings. Moreover, there is a tight relation between encodings and solvers: different encodings are more or less effective depending on the solvers.

Encodings into SAT are constructed every time a new problem is converted into CNF. In this paper we focus on encoding a particular problem, the social golfer problem, studying the effect that encoding decisions have on performance. This work contributes to better understanding the interplay of satisfiability modeling and solving on combinatorial problems.

The rest of the paper is organized as follows. The next section gives some insights on how to encode a problem into SAT. Section 3 describes a particular combinatorial problem: the social golfer problem. Section 4 explains how to encode the social golfer problem into SAT, including how to break symmetries in this highly symmetric problem. Afterwards, experimental results are given for running both a local search and a backtrack search solver (walksat and siege, respectively) for the two encodings of the social golfer problem: one with no symmetry breaking and other with symmetry breaking. Finally, we conclude the paper and suggest future work.

2 Encoding a Problem into SAT

Encoding combinatorial problems as SAT problems has been mostly motivated by the recent advances in SAT solvers. The new solvers are capable of solving very large, very hard real-world problem instances, which more traditional SAT solvers are totally incapable of.

Nonetheless, only a few problems are naturally encoded as SAT problems. Combinational electronic circuits are the most paradigmatic example. Indeed, more sophisticated logics are frequently more adequate to represent most of the problems. Consequently, encoding such problems as CNF formulas may require a significant effort. Hopefully this effort will be counterbalanced by the performance of SAT solvers.

To encode a combinatorial problem into SAT one must define a set of variables and a set of constraints on the variables. Usually we represent SAT problems as CNF formulas, and therefore a formula is a conjunction of clauses, a clause is a disjunction of literals and a literal is a variable or its negation.

The set of variables may be defined based on different criteria: the most intuitive variables set, the set with minimum cardinality, the set that will require the smallest number of clauses, etc. Choosing the most adequate variables *is more an art than a science*. Moreover, the definition of the set of constraints may require the definition of additional auxiliary variables. In some cases, these variables are really essential; in other cases, we prefer to have more variables rather than more clauses.

Recent advances in encodings include identifying and breaking symmetries [Crawford *et al.*, 1996; Brown *et al.*, 1988; Smith, 2001]. There has been a significant effort for studying the effect of symmetry breaking in constraint satisfaction, which has further motivated the study of symmetry breaking in SAT encodings.

Symmetries cause the existence of redundant search paths, which is a clear drawback for backtrack search. Breaking symmetries reduces the search space: this is a clearly advantage for problems having no solution, which implies traversing the whole search space to prove unsatisfiability. For the same reason, breaking symmetries is also an advantage when all the solutions must be found. (Even though symmetrical solutions have to be computed from the solutions found.) Moreover, experimental evaluation has shown that (partially) breaking symmetries can also be useful for finding one solution [Ramani and Markov, 2005]. Observe that with symmetry breaking the freedom of the search is restricted.

On the other hand, there is often a trade-off between the cost of eliminating symmetries and the savings derived from having done so. Complete symmetry breaking make solvers to return a unique solution from each set of symmetrically equivalent ones, which is the one found first by the variable and value ordering heuristics. But usually one is interested in finding any solution as quickly as possible, rather than guaranteeing only distinct solutions are returned.

One may envision three main different ways of eliminating symmetry:

1. Remodel the problem [Smith, 2001]. A different encoding, e.g. obtained by defining a different set of variables, may create a problem with less symmetries.

2. Add constraints to the model [Crawford *et al.*, 1996; Aloul *et al.*, 2003]. Such constraints merge symmetries in equivalent classes. In practice, only one assignment satisfies these constraints, instead of n assignments, where n is the number of elements in a given equivalent class.

3. Change the search process to avoid symmetrically equivalent states [Brown *et al.*, 1988; Gent and Smith, 2000; Fahle *et al.*, 2001]. This can be done by adding constraints to ensure that any assignment symmetric to one assignment already considered will not be explored in the future, or by performing checks that symmetric equivalents have not been visited. This is done for both satisfying and unsatisfying assignments. However, this approach has not found success in SAT. This is unsurprising, because of the reliance of SAT solvers on very small time between branching decisions, limiting the overheads that can be accepted and ruling out these symmetry breaking techniques.

Another approach that aims *reducing* symmetry was proposed by Pedro Meseguer and Carme Torras [Meseguer and Torras, 2001]. The idea is to use symmetry to guide the search. The authors suggest the use of variable and value selection heuristics to direct the search towards subspaces with high density of non-symmetric states.

3 The Social Golfer Problem

The social golfer problem is derived from a question posted to `sci.op-research` in May 1998:

> The coordinator of a local golf club has come to you with the following problem. In her club, there are 32 social golfers, each of whom play golf once a week, and always in groups of 4. She would like you to come up with a schedule of play for these golfers, to last as many weeks as possible, such that no golfer plays in the same group as any other golfer on more than one occasion.

In other words, this problem can be described more explicitly by enumerating four constraints which must be satisfied:

1. The golf club has 32 members.
2. Each member plays golf once a week.
3. Golfers always play in groups of 4.
4. No golfer plays in the same group as any other golfer twice.

Since 1998, this problem has become a famous combinatorial problem. It is problem number 10 in CSPLib (`http://www.csplib.org/`). A solution is said to be optimal when *maximum socialisation* is achieved, i.e. when one golfer plays with as many other golfers as possible. Clearly, since a golfer plays with three new golfers each week, the schedule cannot exceed 10 weeks. This follows from the fact that each golfer plays with three other golfers each week. Since there is a total of 31 other golfers, this means that a golfer runs out of opponents after 31/3 weeks.

For some years, it was not known if a 10 week (and therefore optimal) solution for 32 golfers exists. In 2004, Aguado found a solution using design-theoretic techniques [Aguado, 2004]. No constraint programming technique has yet solved this instance, so it remains a valuable benchmark for the constraint programming community. The best known solution from constraint programming is from Stefano Novello, who posted a 9-week solution, along with the source of the ECL^{i}PS^e program used to find it.

Even though the social golfer problem was described for 32 golfers playing in groups of 4, it can be easily generalized. An instance to the problem is characterized by a triple $w - p - g$, where w is the number of weeks, p is the number of players per group and g is the number of groups. The

week	group 1	group 2
1	1 2	3 4
2	1 3	2 4
3	1 4	2 3

Figure 1: A solution for the social golfer problem 3-2-2.

original question therefore is to find a solution to the w-4-8 problem, with w being the maximum, i.e. to find a solution to 10-4-8 (or prove that none exists). For example, Figure 1 gives a solution for the social golfer problem 3-2-2, i.e. for scheduling 4 golfers playing in 2 groups of 2 golfers each for 3 weeks.

The social golfer problem is related with other well-known combinatorial problems. Indeed, this problem is a generalisation of the problem of constructing a round-robin tournament schedule, the main difference being that in the social golfer problem the number of players in a group may be greater than two. Also, the social golfer problem of finding a 7 week schedule for 5 groups of 3 players (7-3-5) is the same as Kirkman's Schoolgirl Problem, where the main goal is to arrange fifteen schoolgirls in rows of three so that each schoolgirl walks in the same row with every other schoolgirl exactly once a week.

The social golfer problem is also well-known for being a case study of symmetry for constraint programming (e.g. see [Smith, 2001]). This problem is highly symmetric, exhibiting the following symmetries:

- Golfers within a group are interchangeable. Order has no significance for groups of golfers.
- Groups within a week are interchangeable. Again, order has no significance when considering groups within a week.
- Weeks are interchangeable. There are no order constraints with respect to weeks.

The exact group of symmetries that arises from this will depend on the encoding chosen. For example, in the model considered by Harvey, Kelsey and Petrie [Harvey *et al.*, 2003], this gives the wreath product of S_8 with S_{10}. This means that the 8 groupings in each week can be permuted in any way, giving S_8, and that the 10 weeks can also be permuted in any way, giving S_{10}.

Eliminating the above symmetries is not expensive and can bring significant enhancements. For example, considering again the solution given in Figure 1, one may assume that symmetries have been eliminated: this explains why golfers are ordered within groups, groups are ordered within weeks with respect to the first player and weeks are ordered with respect to the second player of the first group.

There is also one final symmetry that is not considered above.

- Golfers are interchangeable. That is, the names of the 32 golfers are insignificant.

In the model just mentioned, the additional symmetry would give a semi-direct product of the previous group with S_{32}. This combination of symmetries makes symmetry breaking much more difficult, and to date no efficient method to break *all* symmetries has been presented. From the very beginning, the social golfer problem has been extensively studied as a paradigmatic problem with a significant number of symmetries [Smith, 2001; Puget, 2002]. In this paper, we concentrate only on the initial group of symmetries of the problem, disregarding the more complicated combination for simplicity. It would certainly be interesting to consider approaches to breaking the full group of the problem, following for example [Aloul *et al.*, 2003], but that is outside the scope of this paper.

4 A SAT Encoding for the Social Golfer Problem

To encode the social golfer problem as a SAT problem we must define:

- A set of variables.
- A set of constraints (represented by clauses) on the variables.

The set of constraints must guarantee that each golfer plays golf once a week, golfers always play in groups of a given size and no golfer plays in the same group as any other golfer twice.

4.1 The Model

We have defined SAT variables based on the golfers. Apparently, for a social golfer problem $w-p-g$ it should be enough to have $w \times (p \times g) \times g$ variables. The value of each variable would allow us to conclude whether, in a given week, a certain golfer is scheduled to play in a particular group.

However, we have chosen a more expressive model. Even though this model has more variables, these variables are quite useful for defining the problem constraints. Instead of $w \times (p \times g) \times g$ variables, this new model has $w \times (p \times g) \times (p \times g)$ variables. When compared with the other model, the difference is that we introduced an additional order relation for golfers within groups. This means that the value of each variable indicates whether golfer i is scheduled to play in group k of week l as the j^{th} player, with $1 \leq i \leq (p \times g)$, $1 \leq j \leq p$, $1 \leq k \leq g$ and $1 \leq l \leq w$. (In what follows we will refer to $x = p \times g$ as the number of golfers.) Although the order of players is irrelevant within groups (as well as the order of groups within weeks and the order of weeks), this model requires most constraints to be at-least-one and at-most-one clauses.

The next step consists in adding clauses to specify that:

- Each golfer plays exactly once per week, i.e.:
  - Each golfer plays at least once per week.
  - Each golfer plays at most once per week.
- Each group in each week has exactly p players, i.e.:
  - At least one golfer must play as the j^{th} golfer, with $1 \leq j \leq p$.
  - At most one golfer can play as the j^{th} golfer, with $1 \leq j \leq p$.

Let us now consider the social golfer problem $w-p-g$, with the number of golfers being given by $x = p \times g$. Consider $\textsc{Golfer}_{ijkl}$ to be a variable equivalent to having golfer i playing as the j^{th} player of group k during week l, with $1 \le i \le x$, $1 \le j \le p$, $1 \le k \le g$ and $1 \le l \le w$.

Each of at-least-one clauses referring to golfers has size $x = p \times g$ and is obtained as simply as follows.

$$\bigwedge_{i=1}^{x} \bigwedge_{l=1}^{w} \bigvee_{j=1}^{p} \bigvee_{k=1}^{g} \textsc{Golfer}_{ijkl}$$

The at-most-one clauses referring to golfers are encoded with two sets of binary clauses. The first set of clauses guarantees that each golfer plays at most once in the same group.

$$\bigwedge_{i=1}^{x} \bigwedge_{l=1}^{w} \bigwedge_{j=1}^{p} \bigwedge_{k=1}^{g} \bigwedge_{m=j+1}^{p} \neg\textsc{Golfer}_{ijkl} \vee \neg\textsc{Golfer}_{imkl}$$

The second set of clauses guarantees that each golfer plays at most once per week.

$$\bigwedge_{i=1}^{x} \bigwedge_{l=1}^{w} \bigwedge_{j=1}^{p} \bigwedge_{k=1}^{g} \bigwedge_{m=k+1}^{g} \bigwedge_{n=j+1}^{p} \neg\textsc{Golfer}_{ijkl} \vee \neg\textsc{Golfer}_{inml}$$

Let us now consider the clauses referring to groups of golfers. Each at-least-one clause has size x and is obtained as follows.

$$\bigwedge_{l=1}^{w} \bigwedge_{k=1}^{g} \bigwedge_{j=1}^{p} \bigvee_{i=1}^{x} \textsc{Golfer}_{ijkl}$$

Finally, the at-most-clauses for groups of golfers are encoded by a set of binary clauses.

$$\bigwedge_{l=1}^{w} \bigwedge_{k=1}^{g} \bigwedge_{j=1}^{p} \bigwedge_{i=1}^{x} \bigwedge_{m=i+1}^{x} \neg\textsc{Golfer}_{ijkl} \vee \neg\textsc{Golfer}_{imkl}$$

With the set of variables and clauses described above we have encoded all the constraints of the problem, except the one that mentions that "no golfer plays in the same group as any other golfer twice". To guarantee this condition, we introduce a set of auxiliary variables and a *ladder* matrix.

The set of auxiliary variables allows us to know exactly which golfers are scheduled to play in each match. Hence, we must have $x \times g \times w$ additional variables. Clearly, the value of these new variables depends on the value of the variables GOLFER described above. Consider these new variables to be a set of variables denoted as $\textsc{Golfer'}_{ikl}$, meaning that golfer i is scheduled to play in group k during week l, with $1 \le i \le x$, $1 \le k \le g$ and $1 \le l \le w$. It is easy to establish an equivalence relation between each variable $\textsc{Golfer'}_{ikl}$ and the corresponding GOLFER variables. (Each equivalence may be readily converted into a set of clauses.)

$$\textsc{Golfer'}_{ikl} \leftrightarrow \bigvee_{j=1}^{p} \textsc{Golfer}_{ijkl}$$

These new variables will now be used by the variables in the ladder matrix in such a way that no golfer plays in the same group as any other golfer more than once.

	1.1	1.2	2.1	2.2	3.1	3.2
3.4	T	**T**	**F**	F	F	F
2.3	T	T	T	T	T	T
2.4	T	T	T	T	F	F
1.2	T	F	F	F	F	F
1.3	T	T	T	F	F	F
1.4	T	T	T	T	T	F

Figure 2: The ladder matrix for the solution given in Figure 1.

The ladder matrix [Gent and Prosser, 2002; Ansótegui and Manyá, 2004; Gent and Nightingale, 2004] is characterized by a set of $\binom{x}{2} \times (g \times w)$ Boolean ladder variables and a set of ladder clauses. Intuitively, one would say that the value of each variable denotes whether two golfers are scheduled to play together in a given group of a given week. But we can do better. We can guarantee that every two golfers play together at most once.

Consider the ladder variables to be denoted as $\textsc{Ladder}_{yz}$, with $1 \le y \le \binom{x}{2}$ and $1 \le z \le g \times w$. A complete assignment of the ladder variables is said to be *valid* if and only if every row is a sequence of zero or more true assignments followed by false assignments. In other words, after a ladder variable being set to FALSE, no subsequent variables in the same row can be assigned TRUE, i.e.:

$$\forall_y \neg\exists_z \bullet \textsc{Ladder}_{yz} = \textsc{False} \wedge \textsc{Ladder}_{yz+1} = \textsc{True}$$

The behavior of the ladder matrix can be used to guarantee that no two golfers play more than once in the same group. Actually, having an adjacent pair of variables with values TRUE and FALSE identifies precisely in which group of which week two golfers played together.

Whenever a ladder variable is satisfied, there is a set of adjacent variables that must be satisfied. This can be achieved by unit propagation adding the following set of clauses.

$$\bigwedge_{y=1}^{\binom{x}{2}-1} \bigwedge_{z=1}^{g\times w} \neg Ladder_{yz+1} \vee Ladder_{yz}$$

For example, consider the solution for the social golfer problem 3-2-2 given in Figure 1. This solution corresponds to the ladder matrix given in Figure 2. Each line in the matrix corresponds to a pair of golfers. For example, the first line named 3.4 indicates when golfers 3 and 4 play together. Each column in the matrix corresponds to a group of golfers. For example, the second column named 1.2 specifies the second group of golfers playing in the first week. Each pair of adjacent entries within a line with values T/F indicate when two golfers play together. For example, the values of the two entries in bold indicate that golfers 3 and 4 play together in the second group of the first week. Observe that due to the ladder matrix constraints no golfer can play with any other golfer more than once.

Finally, the variables in the ladder matrix must be related with the auxiliary variables described above (denoted as GOLFER'). Having $\textsc{Golfer'}_{ikl}$ and $\textsc{Golfer'}_{mkl}$ satisfied means that both golfers i and m play in the same group k in the same week l. This is equivalent to having $Ladder_{[\binom{x-i}{2}+m-i](l\times k)}$ assigned value TRUE and $Ladder_{[\binom{x-i}{2}+m-i](l\times k+1)}$ assigned value FALSE. Formally:

$$\bigwedge_{l=1}^{w}\bigwedge_{k=1}^{g}\bigwedge_{i=1}^{x-1}\bigwedge_{m=i+1}^{x} \neg\text{GOLFER'}_{ikl} \vee \neg\text{GOLFER'}_{mkl} \vee Ladder_{[\binom{x-i}{2}+m-i](l\times k)}$$

$$\bigwedge_{l=1}^{w}\bigwedge_{k=1}^{g}\bigwedge_{i=1}^{x-1}\bigwedge_{m=i+1}^{x} \neg\text{GOLFER'}_{ikl} \vee \neg\text{GOLFER'}_{mkl} \vee \neg Ladder_{[\binom{x-i}{2}+m-i](l\times k+1)}$$

$$\bigwedge_{l=1}^{w}\bigwedge_{k=1}^{g}\bigwedge_{i=1}^{x-1}\bigwedge_{m=i+1}^{x} Ladder_{[\binom{x-i}{2}+m-i](l\times k+1)} \vee \neg Ladder_{[\binom{x-i}{2}+m-i](l\times k)} \vee \neg\text{GOLFER'}_{ikl}$$

$$\bigwedge_{l=1}^{w}\bigwedge_{k=1}^{g}\bigwedge_{i=1}^{x-1}\bigwedge_{m=i+1}^{x} Ladder_{[\binom{x-i}{2}+m-i](l\times k+1)} \vee \neg Ladder_{[\binom{x-i}{2}+m-i](l\times k)} \vee \neg\text{GOLFER'}_{mkl}$$

4.2 Symmetry Breaking

After establishing the model described above, we considered predicates for breaking symmetries in our SAT encoding for the social golfer problem. Clearly, this problem (and therefore our model) is highly symmetric: golfers within a group are interchangeable, groups within a week are also interchangeable and finally weeks are interchangeable. We suggest to tackle these symmetries by adding more clauses to the encoding.

The symmetries between players within the same group are eliminated by forcing players in the same group to be in lexicographic order, i.e. in increasing numerical order. In practice, this is done by adding a set of binary clauses as follows.

$$\bigwedge_{i=1}^{x}\bigwedge_{j=1}^{p}\bigwedge_{k=1}^{g}\bigwedge_{l=1}^{w}\bigwedge_{m=1}^{i-1} \neg\text{GOLFER}_{ijkl} \vee \neg\text{GOLFER}_{m(j+1)kl}$$

These clauses guarantee that if a golfer is scheduled to play as the j^{th} golfer, then the $(j+1)^{th}$ golfer has to be in a higher numerical order.

Similarly, we impose the first players of the groups within the same week to be in lexicographic order. Obviously, golfer #1 must be scheduled as the first golfer in the first group within each week. In addition, we use binary clauses to encode symmetry breaking within each week.

$$\bigwedge_{i=1}^{x}\bigwedge_{k=1}^{g}\bigwedge_{l=1}^{w}\bigwedge_{m=1}^{i-1} \neg\text{GOLFER}_{i1kl} \vee \neg\text{GOLFER}_{m1(k+1)l}$$

These binary clauses impose first golfers of subsequent groups to be in lexicographic order.

Finally, additional clauses are used to break symmetries between weeks. This is simply achieved by imposing lexicographic order between the second golfer of the first group of each week. This is encoded as follows.

$$\bigwedge_{i=1}^{x}\bigwedge_{k=1}^{g}\bigwedge_{l=1}^{w}\bigwedge_{m=1}^{i-1} \neg\text{GOLFER}_{i2kl} \vee \neg\text{GOLFER}_{m2k(l+1)}$$

These three sets of binary clauses suffice to break the symmetries that were initially mentioned. Observe that the solution given in Figure 1 satisfies all the constraints we have specified for symmetry breaking. As we mentioned earlier, we leave for future work the interesting question of how best to tackle the combination of these symmetries with the free interchanging of players.

5 Experimental Results

In this section we evaluate empirically our encodings for the social golfer problem[1]. We compare our encoding with and without symmetries. We use two state-of-the-art SAT solvers: walksat and siege.

Experimental results are given for a set of 29 benchmark problems. All these problems are satisfiable. Otherwise, they would not be solved by local search. Moreover, many of the unsatisfiable problem instances of the social golfer problem are trivially found to be unsatisfiable. On the other hand, it is widely accepted that symmetry breaking helps proving unsatisfiability (e.g. see [Ramani and Markov, 2005]), but not much has been said about finding exactly one solution.

Table 1 characterizes each problem instance (named as $w-p-g$) by giving the number of variables and clauses. The larger instances have thousands of variables and around a million of clauses. We have observed that most of the clauses are either binary or ternary, which makes the average clause size (AvgCS) to be between 2 and 3. We have also observed that the additional clauses for breaking symmetries (SBCls), which are all binary clauses, may augment the number of clauses in the initial model for about 30% for the larger instances (for smaller instances this value is smaller).

5.1 Local Search: Walksat

Walksat [Kautz *et al.*, 2004] is a local search solver. The algorithm is quite simple:

- Start with a random truth assignment.
- With probability p:
  - Pick a variable occurring in some unsatisfied clause and flip its truth assignment.
- With probability $1-p$:
  - Make the best possible local move.
- Repeat the last two steps until the assignment satisfies all clauses.

We have tried to run walksat on our benchmark problems of the social golfer problem. Even though we tried many different configurations, walksat was far from being competitive on solving these problems, specially those including clauses for symmetry breaking. (We also tried other local search

[1]For all experimental results a P-IV@1.7 GHz Linux machine with 1 GByte of physical memory was used.

Problem	# Vars	# Cls	AvgCS	% SBCls
3-2-2	108	446	2.43	9%
5-3-2	495	2547	2.46	10%
4-3-3	864	5598	2.41	17%
7-4-2	1456	8556	2.46	12%
9-5-2	3375	21665	2.45	13%
5-4-4	4000	35032	2.35	24%
11-6-2	6732	46026	2.45	14%
7-6-3	9450	79965	2.38	21%
13-7-2	12103	86751	2.44	15%
6-5-5	13500	147950	2.30	28%
7-7-3	14406	127302	2.38	21%
5-8-3	14880	135780	2.37	22%
3-6-6	15876	207054	2.26	31%
15-8-2	20160	149912	2.44	15%
6-7-4	21756	227402	2.33	26%
3-8-5	24480	303260	2.28	29%
5-7-5	28175	339185	2.29	29%
17-9-2	31671	242541	2.44	16%
4-7-6	32340	440013	2.26	31%
3-9-5	34020	432360	2.28	29%
10-9-3	41310	388341	2.37	22%
6-9-4	43740	484614	2.32	26%
8-10-3	44400	426435	2.37	22%
19-10-2	47500	372630	2.43	16%
3-9-6	48843	701811	2.25	32%
5-10-4	49000	554140	2.32	27%
4-8-7	63616	995876	2.23	34%
5-10-5	76250	991925	2.28	30%
4-9-7	88452	1419075	2.23	34%

Table 1: Social golfer problems: number of variables and clauses.

solvers without success.) This is as suggested by Prestwich, that symmetry breaking constraints reduce the number of solutions and therefore make it harder for local search to find solutions [Prestwich, 2001].

Nonetheless, we have run a problem for a significant number of seeds. Figure 3 compares the average number of flips per second and the total CPU time for including or not including symmetry breaking clauses on the encodings (SymBreak and NoSymBreak, respectively). Results were obtaining running walksat with 1500 seeds for problem 7-4-2. From these results, which we believe to be representative of our SAT benchmark problems of the social golfer problem, we may conclude that:

- Walksat performs more flips per second (in average) without clauses for symmetry breaking. This may be explained by the overhead produced by the additional symmetry breaking clauses.
- Walksat requires more CPU time to solve instances with symmetry breaking clauses.
- Adding clauses to break symmetries affects negatively both the number of flips and the CPU time, although the consequences are more negative for the CPU time. Indeed, for the encoding with symmetry breaking clauses we may observe an extremely fluctuation on the expected time to find a solution, which is probably associated with a heavy-tail distribution [Gomes *et al.*, 2000].

5.2 Backtrack Search: Siege

Siege [Ryan, 2004] is a randomized backtrack search SAT solver enhanced with clause recording. The data structures are carefully implemented and the decision heuristic is very efficient, specially for structured problems.

Siege has been shown to be quite competitive on solving our benchmark problems. We have run siege on each problem for 1500 seeds. Figure 4 compares the number of nodes (median and mean, using a logarithmic scale) for including or not including symmetry breaking clauses. Figure 5 makes the same comparison for the CPU time. Apparently, including symmetry breaking clauses often does not compensate. Furthermore, results for including symmetry breaking clauses are more negative for the number of nodes rather than for the CPU time. The same holds for the median values when compared with the mean values.

With the aim of clarifying the differences between median and mean values, we have run one of the problems where those differences could be observed (problem 6-7-4) for 10000 seeds. Figure 6 gives the number of nodes and the CPU time. From these plots we may conclude that adding symmetry breaking clauses seems to avoid a heavy-tail behavior exhibited by the encoding with no symmetry breaking. Hence, we claim that adding symmetry breaking clauses may avoid the heavy-tail behavior, in particular for the most difficult instances.

6 Conclusions and Future Work

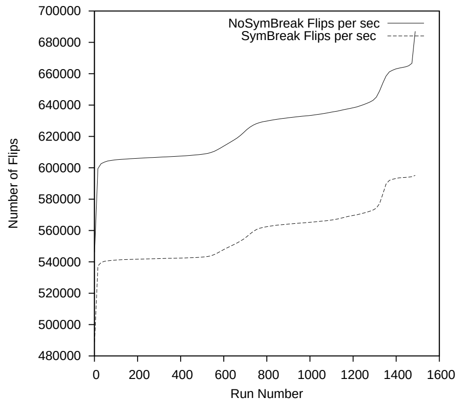
Recent advances in SAT solving motivate an increasing number of combinatorial problems to be encoded into SAT. We argue that modeling decisions have an impact on the solver's performance. We have encoded the social golfer problem into SAT. Two different encodings - with and without symmetry breaking - have been empirically evaluated with local search and backtrack search solvers. A somewhat surprising observation is that some of the encodings, depending on the solvers, may exhibit a heavy-tail distribution. In such circumstances, choosing the *right* encoding can make the difference between heavy-tail behavior or not. In a near future, we plan to do a more comprehensive evaluation, which includes evaluating more instances, trying different encodings and also encoding new problems.

References

[Aguado, 2004] Alejandro Aguado. A 10 days solution to the social golfer problem, 2004. Manuscript.

[Aloul *et al.*, 2003] F. Aloul, K. A. Sakallah, and I. Markov. Efficient symmetry breaking for boolean satisfiability. In *Proc. IJCAI*, pages 271–276, August 2003.

[Ansótegui and Manyá, 2004] Carlos Ansótegui and Felip Manyá. Mapping problems with finite-domain variables into problems with boolean variables. In *Proceedings of*



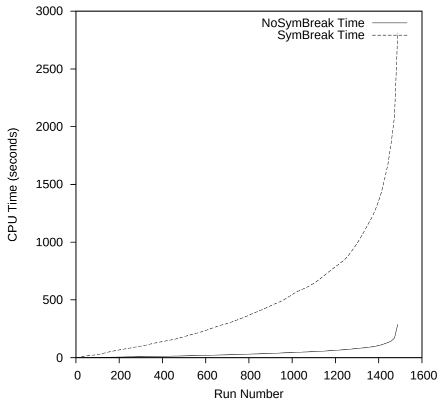


Figure 3: Walksat: average number of flips per second and total CPU time for problem 7-4-2.

the International Conference on Theory and Applications of Satisfiability Testing, May 2004.

[Brown *et al.*, 1988] C. A. Brown, L. Finkelstein, and P. W. Purdom Jr. Backtrack searching in the presence of symmetry. In 6^{th} *International Conference, on Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, volume 357 of *Lecture Notes in Computer Science*, pages 99–110. Springer-Verlag, 1988.

[Crawford *et al.*, 1996] J. M. Crawford, M. L. Ginsberg, E. Luks, and A. Roy. Symmetry-breaking predicates for search problems. In *Proceedings of the International Conference on Principles of Knowledge and Reasoning*, pages 148–159, 1996.

[Een and Sorensson, 2003] N. Een and N. Sorensson. An extensible SAT solver. In *Sixth International Conference on Theory and Applications of Satisfiability Testing*, May 2003.

[Fahle *et al.*, 2001] T. Fahle, S. Schamberger, and M. Sellmann. Symmetry breaking. In *Proc. CP 2001*, pages 93–107, 2001.

[Gent and Nightingale, 2004] Ian P. Gent and Peter Nightingale. A new encoding of alldifferent into sat. In AM Frisch and I Miguel, editors, *Proc. 3rd International Workshop on Modelling and Reformulating Constraint Satisfaction Problems, CP2004*, pages 95–110, 2004.

[Gent and Prosser, 2002] Ian P. Gent and Patrick Prosser. Sat encodings of the stable marriage problem with ties and incomplete lists. In *Fifth International Symposium on Theory and Applications of Satisfiability Testing*, May 2002.

[Gent and Smith, 2000] Ian P. Gent and Barbara M. Smith. Symmetry breaking during search in constraint programming. In *Proc. ECAI*, pages 599–603, 2000.

[Goldberg and Novikov, 2002] E. Goldberg and Y. Novikov. BerkMin: a fast and robust SAT-solver. In *Proceedings of the Design and Test in Europe Conference*, pages 142–149, March 2002.

[Gomes *et al.*, 2000] Carla P. Gomes, Bart Selman, Nuno Crato, and Henry A. Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning*, 24(1/2):67–100, 2000.

[Harvey *et al.*, 2003] Warwick Harvey, Tom Kelsey, and Karen Petrie. Symmetry group generation for CSPs. Technical Report APES-60-2003, APES Research Group, July 2003. Available from http://www.dcs.st-and.ac.uk/~apes/apesreports.html.

[Kautz *et al.*, 2004] Henry Kautz, Bart Selman, and David McAllester. Walksat in the 2004 sat competition. In *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing*, 2004.

[Meseguer and Torras, 2001] P. Meseguer and C. Torras. Exploiting symmetries within constraint satisfaction search. *Artificial Intelligence*, 129(1-2):133–163, 2001.

[Moskewicz *et al.*, 2001] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Engineering an efficient SAT solver. In *Design Automation Conference*, pages 530–535, June 2001.

[Prestwich, 2001] Steven Prestwich. First-solution search with symmetry breaking and implied constraints. In *Workshop on Symmetry in Constraint Satisfaction Problems*, 2001.

[Puget, 2002] Jean-François Puget. Symmetry breaking revisited. In Pascal Van Hentenryck, editor, *Proc. CP 2002*, pages 446–461. Springer-Verlag, 2002.

[Ramani and Markov, 2005] A. Ramani and I. L. Markov. Automatically exploiting symmetries in constraint programming. In *Recent Advances in Constraints*, volume 3419 of *Lecture Notes in Computer Science*, pages 98–112. Springer-Verlag, 2005.

[Ryan, 2004] L. Ryan. Efficient algorithms for clause-learning SAT solvers. Master's thesis, Simon Fraser University, February 2004.

[Smith, 2001] Barbara M. Smith. Reducing symmetry in a combinatorial design problem. In *Proceedings of the Third International Workshop on Integration of AI and OR Techniques*, pages 351–359, 2001.

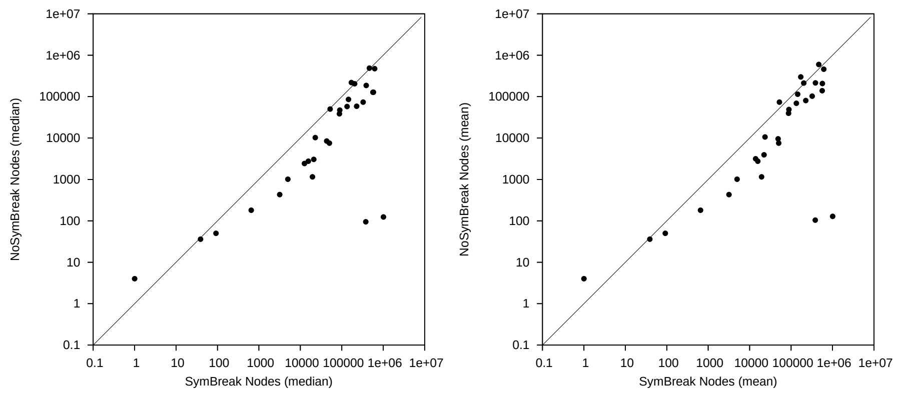


Figure 4: Siege: comparison of the number of nodes (median and mean) for a set of problems.

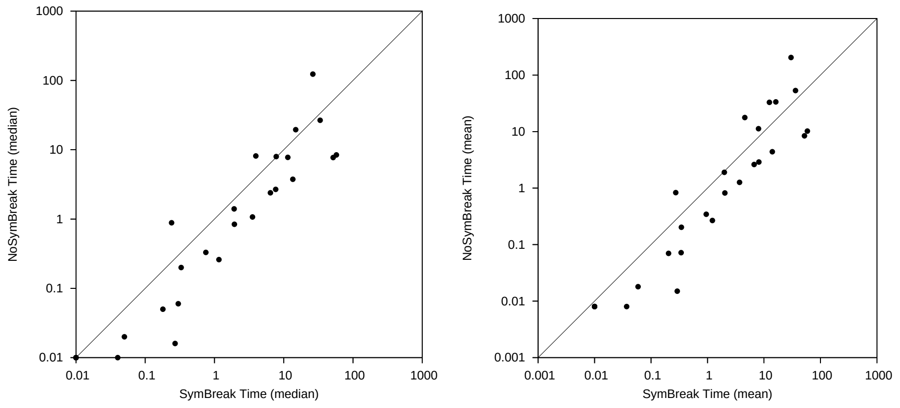


Figure 5: Siege: comparison of the CPU time (median and mean) for a set of problems.

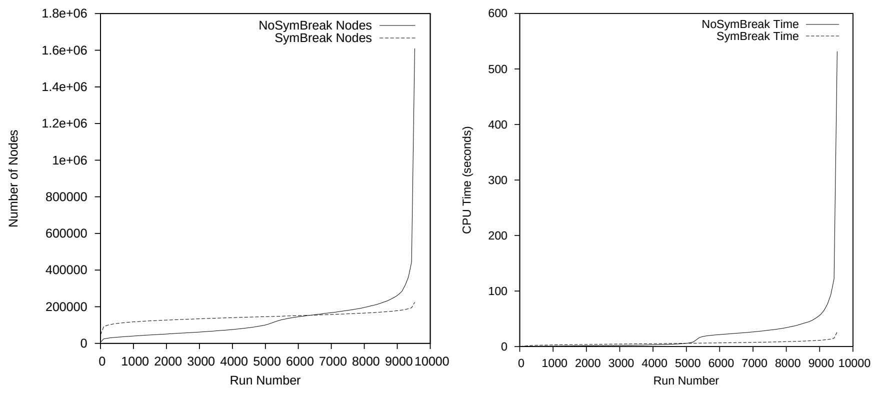


Figure 6: Siege: number of nodes and CPU time for problem 6-7-4.

Modelling and Solving the Stable Marriage Problem Using Constraint Programming

David F. Manlove[*,†] and **Gregg O'Malley**[*]

Department of Computing Science, University of Glasgow, Glasgow G12 8QQ, UK.
Email: {davidm,gregg}@dcs.gla.ac.uk.

Abstract

We study the Stable Marriage problem (SM), which is a combinatorial problem that arises in many practical applications. We present two new models of an instance I of SM with n men and n women as an instance J of a Constraint Satisfaction Problem. We prove that establishing arc consistency in J yields the same structure as given by the established Extended Gale/Shapley algorithm for SM as applied to I. Consequently, a solution (stable matching) of I can be derived without search. Furthermore we show that, in both encodings, *all* stable matchings in I may be enumerated in a failure-free manner. Our first encoding is of $O(n^3)$ complexity and is very natural, whilst our second model, of $O(n^2)$ complexity (which is optimal), is a development of the Boolean encoding in [Gent *et al.*, 2001], establishing a greater level of structure.

1 Introduction

The classical Stable Marriage problem (SM) has been the focus of much attention in the literature over the last few decades [Gale and Shapley, 1962; Knuth, 1976; Gusfield and Irving, 1989; Roth and Sotomayor, 1990]. An instance of SM comprises n men, $m_1, \ldots, m_n$, and n women, $w_1, \ldots, w_n$, and each person has a preference list in which they rank all members of the opposite sex in strict order. A matching M is a bijection between the men and women. We denote the partner in M of a person q by $M(q)$. A (man,woman) pair (m_i, w_j) *blocks* a matching M, or forms a *blocking pair* of M, if m_i prefers w_j to $M(m_i)$ and w_j prefers m_i to $M(w_j)$. A matching that admits no blocking pair is said to be *stable*, otherwise the matching is *unstable*. SM and its variants arise in important practical applications, such as the annual match of graduating medical students to their first hospital appointments in a number of countries (see e.g. [Roth, 1984]).

Gale and Shapley [Gale and Shapley, 1962] showed that every instance I of SM admits a stable matching, and gave an $O(n^2)$ algorithm, linear in the instance size, for finding such a matching in I. A modified version of this algorithm – the Extended Gale/Shapley (EGS) algorithm [Gusfield and Irving, 1989, Section 1.2.4] – avoids some unnecessary steps by deleting from the preference lists certain (man,woman) pairs that cannot belong to a stable matching. Moreover the EGS algorithm aids the development of some useful structural properties of SM [Gusfield and Irving, 1989, Section 1.2.4]. The *man-oriented* version of the EGS algorithm (henceforth referred to as the MEGS algorithm) involves a sequence of proposals from the men to the women, provisional engagements between men and women, and deletions from the preference lists. A pseudocode description of MEGS algorithm is given in Figure 1 (the term *delete the pair* (p, w) means that p should be deleted from w's list and vice versa.) The stable matching returned by the MEGS algorithm is called the *man-optimal* (or equivalently, *woman-pessimal*) stable matching, denoted by M_0, since each man has the best partner (according to his ranking) that he could obtain, whilst each woman has the worst partner that she could obtain, in any stable matching. A similar proposal sequence from the women to the men yields the *woman-oriented* EGS (WEGS) algorithm. This gives rise to the *woman-optimal* (or *man-pessimal*) stable matching, denoted by M_z, with analogous properties.

Upon termination of the MEGS algorithm, the reduced preference lists that arise following the deletions are referred to as the *MGS-lists*. Similarly, the *WGS-lists* arise upon termination of the WEGS algorithm. The intersection of the MGS-lists with the WGS-lists yields the *GS-lists* [Gusfield and Irving, 1989, p.16]. Some important structural properties of the GS-lists are given by the following theorem.

Theorem 1 ([Gusfield and Irving, 1989, Theorem 1.2.5]). *For a given instance of SM:*

(i) all stable matchings are contained in the GS-lists;

(ii) no matching M contained in the GS-lists can be blocked by a pair that is not in the GS-lists;

(iii) in the man-optimal (respectively woman-optimal) stable matching, each man is partnered by the first (respectively last) woman on his GS-list, and each woman by the last (respectively first) man on hers.

[*]Supported by Engineering and Physical Sciences Research Council grant GR/R84597/01.
[†]Supported by Royal Society of Edinburgh/Scottish Executive Personal Research Fellowship.

```
assign each person to be free;
while some man $m$ is free and $m$ has a nonempty list loop
    $w$ := first woman on $m$'s list;    {$m$ 'proposes' to $w$}
    if some man $p$ is engaged to $w$ then
        assign $p$ to be free;
    end if;
    assign $m$ and $w$ to be engaged to each other;
    for each successor $p$ of $m$ on $w$'s list loop
        delete the pair $(p, w)$;
    end loop;
end loop;
```

Figure 1: The man-oriented Extended Gale/Shapley algorithm for SM and SMI.

An example SM instance I is given in Figure 2. (We assume that a person's preference list is ordered with his/her most-preferred partner leftmost.) This figure also indicates those preference list entries that belong to the GS-lists. In I, the man-optimal stable matching M_0 and the woman-optimal stable matching M_z are as follows:

$$\begin{aligned} M_0 &= \{(m_1, w_1), (m_2, w_3), (m_3, w_2), (m_4, w_4)\} \\ M_z &= \{(m_1, w_3), (m_2, w_1), (m_3, w_4), (m_4, w_2)\}. \end{aligned}$$

The extension SMI of SM arises when preference lists may be incomplete. This occurs when a person may find a member of the opposite sex *unacceptable*. If a person p finds a person q unacceptable, q does not appear on the preference list of p. In the SMI case, a matching M in an instance I of SMI is a one-one correspondence between a subset of the men and a subset of the women, such that $(m, w) \in M$ implies that each of m and w finds the other acceptable. Given a matching M in an SMI instance, a pair (m, w) blocks a matching M if each of m and w finds the other acceptable, and each is either unmatched in M or prefers the other to their partner in M. If a person p finds a person q unacceptable, then p and q cannot be paired in any stable matching, nor can they form a blocking pair. Hence, from the point of view of finding stable matchings, we lose no generality by assuming that q finds p unacceptable also, so that preference lists are *consistent*. It is straightforward to adapt the EGS algorithm to the SMI case [Gusfield and Irving, 1989, Section 1.4.2] – see Figure 1 for a pseudocode description. The woman-oriented algorithm is analogous. In the SMI context a stable matching need not be complete; however the same set of men and women are matched in all stable matchings [Gale and Sotomayor, 1985]. Furthermore, the concept of GS-lists can be extended to SMI, with analogous properties (for Property (ii) in Theorem 1, each person with a non-empty GS-list should be matched in M; for Property (iii), each person with an empty GS-list is unmatched in both stable matchings).

Men's lists	Women's lists
m_1: w_2 w_4 $\underline{w_1}$ $\underline{w_3}$	w_1: $\underline{m_2}$ $\underline{m_4}$ m_3 $\underline{m_1}$
m_2: $\underline{w_3}$ w_4 $\underline{w_1}$ w_2	w_2: $\underline{m_4}$ $\underline{m_3}$ m_1 m_2
m_3: $\underline{w_2}$ $\underline{w_4}$ w_1 w_3	w_3: m_3 m_4 $\underline{m_1}$ $\underline{m_2}$
m_4: $\underline{w_4}$ $\underline{w_1}$ $\underline{w_2}$ w_3	w_4: $\underline{m_3}$ $\underline{m_4}$ m_2 m_1

Figure 2: An SM instance with 4 men and 4 women; preference list entries that belong to the GS-lists are underlined.

1.1 Related work

The Stable Marriage problem has its roots as a combinatorial problem, but has also been the subject of much interest from the Game Theory and Economics community [Roth and Sotomayor, 1990] and the Operations Research community [Vate, 1989]. In recent years SM and SMI have also been the focus of interest from the Constraint Programming community [Aldershof and Carducci, 1999; Dye, 2001; Gent *et al.*, 2001; Lustig and Puget, 2001; Gent and Prosser, 2002a; 2002b; Green and Cohen, 2003; Thorn, 2003]. These papers have presented a range of encodings of SM and its variants as an instance of a Constraint Satisfaction Problem (CSP). In all references apart from [Gent *et al.*, 2001], structural relationships between the effect of Arc Consistency (AC) propagation [Bessière and Régin, 1997] and the GS-lists were not explored in detail, nor did the authors consider the aspect of failure-free enumeration.

However such issues were considered by Gent et al. [Gent *et al.*, 2001], who proposed two CSP encodings of SMI. For each model, it was shown that AC propagation can be used to achieve similar results to the EGS algorithm in a certain sense. The first encoding creates a CSP instance J_1 using a set of 'conflict matrices' to encode an SMI instance I. In J_1, AC may be established in $O(n^4)$ time, following which the variables' domains correspond to the GS-lists of I. The second encoding creates a Boolean CSP instance J_2. In J_2, AC may be established in $O(n^2)$ time, however the variables' domains after AC propagation only correspond to a weaker structure called the *XGS-lists* in I, which in general are supersets of the GS-lists in I. (The XGS-list for a person p consists of all entries in p's preference list between the first and last entries of his/her GS-list inclusive.) In both encodings the set of all stable matchings in I can be enumerated in a failure-free manner (using a value-ordering heuristic in the case of the first encoding).

1.2 Our contribution

The work of [Gent *et al.*, 2001] left open the question as to whether there exists an $O(n^2)$ CSP encoding of SM that captures exactly the structure of the GS-lists. In this paper we present two encodings of an instance I of SMI (and so of SM) as a CSP instance J. Again, for each encoding, we show that AC propagation achieves the same results as the EGS algorithm in a precise sense. The first model is a natural $(n + 1)$-valued encoding of SMI; it bears some resemblance to the encoding of SM given in [Lustig and Puget, 2001] and develops the 'conflict matrices' model of [Gent *et al.*, 2001]. In this model we show that AC propagation may be carried out in $O(n^3)$ time. Our model is more intuitive, and is more time and space-efficient, than the 'conflict matrices' model. Our second model is a more compact 4-valued encoding that develops the Boolean encoding from [Gent *et al.*, 2001] – in this case we show that AC propagation may be carried out in $O(n^2)$ time. For both models we prove that the GS-lists in I correspond to the domains remaining after establishing AC in J. Furthermore, we show that, for both encodings, we are guaranteed a failure-free enumeration of all stable matchings in I using AC propagation combined with a value-ordering heuristic in J. Our second encoding therefore answers the question left open by [Gent *et al.*, 2001].

Our results show that, provided the model is chosen carefully, AC propagation within a CSP formulation of SMI captures the structure produced by the EGS algorithm. Moreover our second encoding indicates that AC propagation can be achieved within the same time complexity as the (optimal) MEGS algorithm for SMI, producing equivalent structural results. This strengthens the assertion in [Gent *et al.*, 2001] regarding the applicability of constraint programming to the general domain of stable matching problems. Furthermore, in many practical situations there may be additional constraints that cannot be accommodated by a straightforward modification of the EGS algorithm. Such constraints could however be built on top of either of the two models that we present here. Possible extensions could arise from variants of SMI that are NP-hard [Ronn, 1990; Ng and Hirschberg, 1991; Kato, 1993; Manlove *et al.*, 2002].

We remark that, independently, Unsworth and Prosser have formulated a specialised n-ary constraint for SMI, such that AC propagation gives rise to the GS-lists, where the complexity of establishing AC is $O(n^2)$ [Unsworth and Prosser, 2005a]. They have also constructed a specialised binary constraint for SMI that yields the same structure, where AC may be established in $O(n^3)$ time [Unsworth and Prosser, 2005b]. In both cases, all stable matchings may be generated using a failure-free enumeration.

The remainder of this paper is organised as follows. Section 2 contains the $(n+1)$-valued encoding. We show that AC may be established in $O(n^3)$ time, proving the structural relationship between AC propagation and the GS-lists. This is followed by the failure-free enumeration result for this model. In Section 3 we present the 4-valued encoding, following a similar approach, however in this case we show that AC may be established in $O(n^2)$ time. Finally, Section 4 contains some concluding remarks.

2 $(n+1)$-valued encoding

2.1 Overview of the encoding

In this section we present an $(n+1)$-valued binary CSP encoding for an instance I of SMI. We assume that $\mathcal{M} = \{m_1, m_2, \ldots, m_n\}$ is the set of men and $\mathcal{W} = \{w_1, w_2, \ldots, w_n\}$ is the set of women in I (it is not difficult to extend our encoding to the case that the numbers of men and women are not equal, but for simplicity we assume that they are equal). For each man $m_i \in \mathcal{M}$ and woman $w_j \in \mathcal{W}$, the length of m_i's and w_j's preference list is denoted by l_i^m and l_j^w respectively. We let L denote the total length of the preference lists in I. Also, for any person $z \in \mathcal{M} \cup \mathcal{W}$, we let $PL(z)$ denote the set of persons on z's original preference list in I, and we let $GS(z)$ denote the set of persons on z's GS-list in I. For each man $m_i \in \mathcal{M}$ and woman $w_j \in PL(m_i)$, we denote the position of w_j on m_i's original preference list (regardless of any deletions that may be carried out by the MEGS/WEGS algorithms) by $rank(m_i, w_j)$, with $rank(w_j, m_i)$ being similarly defined. If $w_j \in \mathcal{W} \backslash PL(m_i)$, then $rank(m_i, w_j)$ and $rank(w_j, m_i)$ are undefined.

We define a CSP encoding J for an instance I of SMI by introducing $2n$ variables to represent the men and women in the original instance I. For each man $m_i \in \mathcal{M}$, we introduce a variable x_i in J whose domain, denoted by $dom(x_i)$, is initially defined as $dom(x_i) = \{1, 2, \ldots, l_i^m\} \cup \{n+1\}$. Similarly, for each woman $w_j \in \mathcal{W}$, we introduce a variable y_j in J whose domain, denoted by $dom(y_j)$, is initially defined as $dom(y_j) = \{1, 2, \ldots, l_j^w\} \cup \{n+1\}$.

An intuitive meaning of the variables is now given. Informally, if $x_i = p$ ($1 \leq p \leq l_i^m$), then m_i marries the woman w_j such that $rank(m_i, w_j) = p$, and similarly for the case that $y_j = q$ ($1 \leq q \leq l_j^w$). More formally, if $\min dom(x_i) \geq p$ ($1 \leq p \leq l_i^m$), then the pair (m_i, w_l) has been deleted as part of the MEGS algorithm applied to I, for all w_l such that $rank(m_i, w_l) < p$. Hence if w_j is the woman such that $rank(m_i, w_j) = p$, then either m_i proposes to w_j during the execution of the MEGS algorithm or the pair (m_i, w_j) will be deleted before the proposal occurs. Similarly if $\min dom(y_j) \geq q$ ($1 \leq q \leq l_j^w$), then the pair (m_k, w_j) has been deleted as part of the WEGS algorithm applied to I, for all m_k such that $rank(m_k, w_j) < q$. Hence if m_i is the man such that $rank(w_j, m_i) = q$, then either w_j proposes to m_i during the execution of the WEGS algorithm or the pair (m_i, w_j) will be deleted before the proposal occurs. If $x_i = n+1$ (respectively $y_j = n+1$) then m_i (respectively w_j) is unmatched upon termination of each of the MEGS or WEGS algorithms applied to I.

The constraints used for the $(n+1)$-valued encoding are shown in Figure 3. In the context of Constraints 1 and 4, j is the integer such that $rank(m_i, w_j) = p$; also $q = rank(w_j, m_i)$. In the context of Constraints 2 and 3, i is the integer such that $rank(w_j, m_i) = q$; also $p = rank(m_i, w_j)$.

An interpretation of Constraints 1 and 3 is now given (a similar interpretation can be attached to Constraints 2 and 4 with the roles of the men and women reversed). First consider Constraint 1, a stability constraint. This ensures that if a man m_i obtains a partner no better than his p^{th}-choice woman w_j, then w_j obtains a partner no worse than her q^{th}-choice man m_i. Now consider Constraint 3, a consistency constraint. This ensures that if man m_i is removed from w_j's list, then w_j is removed from m_i's list.

2.2 Arc consistency in the $(n+1)$-valued encoding

We now show that, given the above CSP encoding J of an SMI instance I, the domains of the variables in J following AC propagation correspond to the GS-lists of I. That is, we prove that, after AC is established, for any i, j ($1 \leq i, j \leq n$), $w_j \in GS(m_i)$ if and only if $p \in dom(x_i)$, and similarly $m_i \in GS(w_j)$ if and only if $q \in dom(y_j)$, where $rank(m_i, w_j) = p$ and $rank(w_j, m_i) = q$.

1.	$x_i \geq p \Rightarrow y_j \leq q$	$(1 \leq i \leq n, 1 \leq p \leq l_i^m)$
2.	$y_j \geq q \Rightarrow x_i \leq p$	$(1 \leq j \leq n, 1 \leq q \leq l_j^w)$
3.	$y_j \neq q \Rightarrow x_i \neq p$	$(1 \leq j \leq n, 1 \leq q \leq l_j^w)$
4.	$x_i \neq p \Rightarrow y_j \neq q$	$(1 \leq i \leq n, 1 \leq p \leq l_i^m)$

Figure 3: The constraints for the $(n+1)$-valued encoding of an instance SMI.

The proof is presented using two lemmas. The first lemma shows that the arc consistent domains are equivalent to subsets of the GS-lists. This is done by proving that the deletions made by the MEGS and WEGS algorithms applied to I are correspondingly made during AC propagation. The second lemma shows that the GS-lists correspond to a subset of the domains remaining after AC propagation. This is done by proving that the GS-lists for I give rise to arc consistent domains for the variables in J.

Lemma 2. *For a given i $(1 \leq i \leq n)$, let p be an integer $(1 \leq p \leq l_i^m)$ such that $p \in dom(x_i)$ after AC propagation. Then the woman w_j such that $rank(m_i, w_j) = p$ belongs to the GS-list of m_i. A similar correspondence holds for the women.*

Proof. The GS-lists are constructed as a result of the deletions made by the MEGS and WEGS algorithms applied to I. We show that the corresponding deletions are made to the relevant variables' domains during AC propagation. In the following proof, only deletions made by the MEGS algorithm are considered; a similar argument can be used to prove the result for an execution of the WEGS algorithm.

We prove the following fact by induction on the number of proposals z during an execution E of the MEGS algorithm. If proposal z consists of man m_i proposing to woman w_j, with $rank(m_i, w_j) = p$ and $rank(w_j, m_i) = q$, then $x_i \geq p$, $y_j \leq q$ and for each man m_k such that $rank(w_j, m_k) = s$ $(q < s \leq l_j^w)$, $x_k \neq r$, where $rank(m_k, w_j) = r$.

First consider the base case where $z = 1$. Then $p = 1$. Since $x_i \geq 1$, propagation of Constraint 1 yields $y_j \leq q$. Then for each s $(q < s \leq l_j^w)$, propagation of Constraint 3 gives $x_k \neq r$ where $rank(w_j, m_k) = s$ and $rank(m_k, w_j) = r$.

Now suppose that $z = c > 1$ and that the result holds for $z < c$. We consider the cases where $p = 1$ and $p > 1$.
Case (i). For $p = 1$ the proof is similar to that of the base case.
Case (ii). Now suppose that $p > 1$. Let w_l be any woman such that $rank(m_i, w_l) = r < p$. Then w_l has been deleted from m_i's list during the MEGS algorithm. Now suppose $rank(w_l, m_i) = s_1$. Then m_i was deleted from w_l's preference list because she received a proposal from a man m_k whom she prefers to m_i, where $rank(w_l, m_k) = s_2 < s_1$. Since m_k proposed to w_l before the c^{th} proposal, we have by the induction hypothesis that $y_l \leq s_2$, so that $y_l \neq s_1$ and $x_i \neq r$. But w_l was arbitrary and hence $x_i \neq r$ for $1 \leq r \leq p-1$, so that $x_i \geq p$. The rest of the proof is similar to that of the base case. □

Lemma 3. *For each i $(1 \leq i \leq n)$, define a domain of values $dom(x_i)$ for the variable x_i as follows: if $GS(m_i) = \emptyset$, then $dom(x_i) = \{n+1\}$; otherwise $dom(x_i) = \{rank(m_i, w_j) : w_j \in GS(m_i)\}$. The domain of each y_j $(1 \leq j \leq n)$ is defined analogously. Then the domains so defined are arc consistent in J.*

Proof. To show that the variables' domains are arc consistent we consider each constraint in turn.

First consider Constraint 1 and suppose that $x_i \geq p$. Then during the execution of the MEGS algorithm applied to I, either (i) m_i proposed to w_j, or (ii) the pair (m_i, w_j) was deleted, where $rank(m_i, w_j) = p$ and $rank(w_j, m_i) = q$. We consider the two cases below:
Case (i) If m_i proposed to w_j during the execution of the MEGS algorithm, then all men ranked below m_i on w_j's list are deleted, i.e. $y_j \leq q$ as required.
Case (ii) If (m_i, w_j) was deleted during the execution of the MEGS algorithm then w_j must have received a proposal from a man m_k whom she prefers to m_i, where $rank(w_j, m_k) = s$ $(s < q)$. Therefore the MEGS algorithm deletes all those men m_z from w_j's list such that $rank(w_j, m_z) > s$, i.e. $y_j \leq s < q$ as required.

Next consider Constraint 3. Suppose that $y_j \neq q$, so that during an execution of either the MEGS or WEGS algorithms, m_i is deleted from w_j's list, where $rank(w_j, m_i) = q$. To ensure that the preference lists are consistent, the same algorithm deletes w_j from m_i's list, i.e. $x_i \neq p$, where $rank(m_i, w_j) = p$, as required.

Verifying Constraints 2 and 4 is similar to the above with the roles of the men and women reversed and the MEGS algorithm exchanged for the WEGS algorithm. □

The two lemmas above, together with the fact that AC algorithms find the unique maximal set of arc consistent domains, lead to the following theorem.

Theorem 4. *Let I be an instance of SMI, and let J be a CSP instance obtained by the (n+1)-valued encoding. Then the domains remaining after AC propagation in J correspond to the GS-lists of I in the following sense: for any i, j $(1 \leq i, j \leq n)$, $w_j \in GS(m_i)$ if and only if $p \in dom(x_i)$, and similarly $m_i \in GS(w_j)$ if and only if $q \in dom(y_j)$, where $rank(m_i, w_j) = p$ and $rank(w_j, m_i) = q$.*

The constraints shown in Figure 3 may be revised in $O(1)$ time during propagation, assuming that upper and lower bounds for the variables' domains are maintained. Hence the time complexity for establishing AC is $O(ed)$, where e is the number of constraints and d is the domain size [van Hentenryck *et al.*, 1992]. For this encoding we have $e = O(n^2)$ and $d = O(n)$, therefore AC may be established in $O(n^3)$ time; also the space complexity is $O(L)$. These complexities represent an improvement on the 'conflict matrices' encoding in [Gent *et al.*, 2001], whose time and space complexities are $O(n^4)$ and $O(L^2)$ respectively. Moreover we claim that the model that we present in this section is a very natural and intuitive encoding for SMI.

Theorems 4 and 1(iii) show that we can find a solution to the CSP giving the man-optimal stable matching M_0 without search: for each man $m_i \in \mathcal{M}$, we let $p = \min dom(x_i)$. If $p = n+1$ then m_i is unmatched in M_0, otherwise the partner of m_i is the woman $w_j \in \mathcal{W}$ such that $rank(m_i, w_j) = p$. Considering the y_j variables in a similar fashion gives the woman-optimal stable matching M_z.

In fact we may go further and show that the CSP encoding yields all stable matchings in I without having to backtrack due to failure. That is, we may enumerate all solutions of I in a failure-free manner using AC propagation in J combined with a value-ordering heuristic. The following theorem, proved in [Manlove and O'Malley, 2005], describes the enumeration procedure.

Theorem 5. *Let I be an instance of SMI and let J be a CSP instance obtained using the $(n+1)$-valued encoding. Then the following search process enumerates all solutions in I without repetition and without ever failing due to an inconsistency:*

- *AC is established as a preprocessing step, and after each branching decision, including the decision to remove a value from a domain;*
- *if all domains are arc consistent and some variable x_i has two or more values in its domain, then the search proceeds by setting x_i to the minimum value p in its domain. On backtracking, the value p is removed from the domain of x_i;*
- *when a solution is found, it is reported and backtracking is forced.*

3 4-valued encoding

3.1 Overview of the encoding

In this section we present a CSP encoding of SMI that is more complex but more efficient than the $(n+1)$-valued encoding given in Section 2.1. We assume the notation as defined for an instance of SMI in the first paragraph of Section 2.1.

We construct a CSP encoding J for an SMI instance I by introducing L variables, each of which represents a preference list entry. For each man m_i $(1 \leq i \leq n)$ we introduce l_i^m variables $x_{i,p}$ $(1 \leq p \leq l_i^m)$, corresponding to the members of $PL(m_i)$. Similarly for each woman w_j $(1 \leq j \leq n)$ we introduce l_j^w variables $y_{j,q}$ $(1 \leq q \leq l_j^w)$. As before the domain of a variable z is denoted by $dom(z)$; initially each variable is given the domain $\{0,1,2,3\}$.

An intuitive meaning of the variables' values is given in Figure 4. The table indicates that deletions carried out by the MEGS and WEGS algorithms applied to I are reflected by the removal of elements from the relevant variables' domains. In particular, removal of the value 2 (respectively 3) from a variable's domain corresponds to a preference list entry being deleted by the MEGS (respectively WEGS) algorithm applied to I. Note that potentially a given preference list entry could be deleted by both algorithms. Also, if the value 0 is removed from $dom(x_{i,p})$ $(1 \leq i \leq n$, $1 \leq p \leq l_i^m)$, then either m_i proposes to w_j during the MEGS algorithm (where $rank(m_i, w_j) = p$) or the entry is deleted prior to the proposal occurring. Similarly if the value 0 is removed from $dom(y_{j,q})$ $(1 \leq j \leq n$, $1 \leq q \leq l_j^w)$, then either w_j proposes to m_i during the WEGS algorithm (where $rank(w_j, m_i) = q$) or the entry is deleted prior to the proposal occurring.

The constraints for this encoding are listed in Figure 5. In the context of Constraints 4 and 10, j is the integer such that $rank(m_i, w_j) = p$; also $q = rank(w_j, m_i)$. In the context of Constraints 5 and 9, i is the integer such that $rank(w_j, m_i) = q$; also $p = rank(m_i, w_j)$. Further, we remark that Constraints 4 and 9 are present only if $q+1 \leq l_j^w$ and $p+1 \leq l_i^m$ respectively.

An interpretation of each constraint is now given. Firstly consider Constraint 1. This constraint is used to start the proposal sequence and can be interpreted as each man initially proposing to the first woman on his list during the MEGS algorithm. Constraint 2 states that if (m_i, w_l) has been deleted by the MEGS algorithm for all w_l such that $rank(m_i, w_l) < p$, and (m_i, w_j) has also been deleted, where $rank(m_i, w_j) = p$, then (m_i, w_l) has been deleted by the by MEGS algorithm for all w_l such that $rank(m_i, w_l) \leq p$. Hence, if $p+1 \leq l_i^m$, m_i will subsequently propose to the woman w_l such that $rank(m_i, w_l) = p+1$ during the MEGS algorithm, or the pair (m_i, w_l) will be deleted before the proposal occurs. Constraint 3 states that if a woman's q^{th}-choice partner is deleted during an iteration of the MEGS algorithm, then her $(q+1)^{th}$-choice partner should also be deleted. Constraint 4 shows a stability constraint: this ensures that if man m_i obtains a partner no better than w_j, then w_j obtains a partner no worse than m_i. Lastly Constraint 5 is a consistency constraint: this ensures that if m_i is removed from w_j's list during the MEGS algorithm then w_j is also removed from m_i's list. Constraints 6-10 have a similar meaning with the roles of the men and women reversed, and with MEGS replaced by WEGS.

3.2 Arc consistency in the 4-valued encoding

We now prove that, given the above CSP encoding J of an SMI instance I, the domains of the variables in J following AC propagation correspond to the GS-lists of I. That is, we show that, after AC is established, for any i, j $(1 \leq i, j \leq n)$, $w_j \in GS(m_i)$ if and only if $\{2,3\} \subseteq dom(x_{i,p})$, and similarly $m_i \in GS(w_j)$ if and only if $\{2,3\} \subseteq dom(y_{j,q})$, where $rank(m_i, w_j) = p$ and $rank(w_j, m_i) = q$.

In order to establish this correspondence, we define the *GS-domains* for the variables in J as follows. Initially let each variable in J have domain $\{0,1,2,3\}$. Run the MEGS algorithm on instance I. Then use rules (i), (ii) and (v) in Figure 4 to remove 0's and 2's from the appropriate domains, obtaining CSP instance J' from J. Next run the WEGS algorithm on the original instance I. Now use rules (iii), (iv) and (vi) in Figure 4 to remove 0's and 3's from the appropriate domains in J', obtaining CSP instance J''. The domains of the variables in J'' are referred to as the *GS-domains*.

As in Section 2.2, two lemmas are used to prove that enforcing AC gives the GS-lists. The first lemma shows that the domains remaining following AC propagation are equivalent to subsets of the GS-lists. This is done by proving that if a deletion is made as part of either the MEGS or WEGS algorithms, then a corresponding deletion is made during AC propagation. The second lemma shows that the GS-lists correspond to a subset of the domains remaining after AC is enforced. This is done by proving that the GS-domains for J are arc consistent.

Lemma 6. *For a given i $(1 \leq i \leq n)$, let p be an integer such that $\{2,3\} \subseteq dom(x_{i,p})$ after AC propagation. Then the woman w_j such that $rank(m_i, w_j) = p$ belongs to the GS-list of m_i. A similar correspondence holds for the women.*

Proof. The GS-lists are obtained through deletions made by the MEGS and WEGS algorithms. We prove that the corresponding deletions are made to the relevant variables' domains during AC propagation. In particular, suppose that

(i)	$0 \notin dom(x_{i,p})$	$\Leftrightarrow$	$p = 1$ or $2 \notin dom(x_{i,r})$ for all r $(1 \leq r < p)$ (i.e. man m_i's r^{th}-choice woman is removed from his list as part of the MEGS algorithm applied to I, for all r $(1 \leq r < p)$);
(ii)	$2 \notin dom(x_{i,p})$	$\Leftrightarrow$	man m_i's p^{th}-choice woman is removed from his list as part of the MEGS algorithm applied to I;
(iii)	$3 \notin dom(x_{i,p})$	$\Leftrightarrow$	man m_i's p^{th}-choice woman is removed from his list as part of the WEGS algorithm applied to I;
(iv)	$0 \notin dom(y_{j,q})$	$\Leftrightarrow$	$q = 1$ or $3 \notin dom(y_{i,s})$ for all s $(1 \leq s < q)$ (i.e. woman w_j's s^{th}-choice man is removed from her list as part of the WEGS algorithm applied to I, for all s $(1 \leq s < q)$);
(v)	$2 \notin dom(y_{j,q})$	$\Leftrightarrow$	woman w_j's q^{th}-choice man is removed from her list as part of the MEGS algorithm applied to I;
(vi)	$3 \notin dom(y_{j,q})$	$\Leftrightarrow$	woman w_j's q^{th}-choice man is removed from her list as part of the WEGS algorithm applied to I.

Figure 4: Intuitive variable meanings for the 4-valued SMI encoding.

$m_i \in \mathcal{M}$ and $w_j \in PL(m_i)$. Let $p = rank(m_i, w_j)$ and $q = rank(w_j, m_i)$. Then we prove:

– (m_i, w_j) deleted during MEGS algorithm $\Leftrightarrow$ $x_{i,p} \neq 2$ and $y_{j,q} \neq 2$.

– (m_i, w_j) deleted during WEGS algorithm $\Leftrightarrow$ $x_{i,p} \neq 3$ and $y_{j,q} \neq 3$.

In this proof, only deletions made by the MEGS algorithm are considered; a similar argument can be used for deletions made by the WEGS algorithm.

It suffices to prove the following by induction on the number of proposals z during an execution E of the MEGS algorithm. If proposal z consists of man m_i proposing to woman w_j, with $rank(m_i, w_j) = p$ and $rank(w_j, m_i) = q$, then $x_{i,p} > 0$, $y_{j,s} \neq 2$ $(q < s \leq l_j^w)$, and for each man m_k such that $rank(w_j, m_k) = s$ $(q < s \leq l_j^w)$, $x_{k,r} \neq 2$, where $rank(m_k, w_j) = r$.

First consider the base case where $z = 1$. Then $p = 1$. By Constraint 1, $x_{i,1} > 0$, and by Constraint 4 we have $y_{j,q+1} \neq 2$. Hence by Constraint 3, it follows that $y_{j,s} \neq 2$ for each s $(q < s \leq l_j^w)$. Also for each such s, propagation of Constraint 5 ensures that $x_{k,r} \neq 2$, where $rank(w_j, m_k) = s$ and $rank(m_k, w_j) = r$.

Now suppose that $z = c > 1$ and that the result holds for $z < c$. We consider the cases where $p = 1$ and $p > 1$.

Case (i) For $p = 1$ the proof is similar to that of the base case.

Case (ii) Now assume that $p > 1$. Let w_l be any woman such that $rank(m_i, w_l) = r < p$. Then w_l has been deleted from m_i's list during the MEGS algorithm. Now suppose that $rank(w_l, m_i) = s_1$. Then m_i was deleted from w_l's list because she received a proposal from a man m_k whom she prefers to m_i, where $rank(w_l, m_k) = s_2 < s_1$. Since m_k proposed to w_l before the c^{th} proposal, by the induction hypothesis it follows that $x_{i,r} \neq 2$. However since w_l was arbitrary, it follows that $x_{i,r} \neq 2$ for $1 \leq r \leq p - 1$. From Constraint 1 we have $x_{i,1} > 0$, and hence the propagation of Constraint 2 ($p - 1$ times) yields $x_{i,p} > 0$. The rest of the proof is similar to that of the base case. □

Lemma 7. *The GS-domains (corresponding to the GS-lists in I) are arc consistent in J.*

Proof. We consider each constraint in turn to show that the GS-domains are arc consistent.

Clearly Constraint 1 is satisfied, as $p = 1$ in rule (i) of Figure 4, i.e. $x_{i,1} > 0$. Now consider Constraint 4 and suppose that $x_{i,p} > 0$. Then during the execution of the MEGS algorithm, either (i) m_i proposed to w_j, or (ii) the pair (m_i, w_j) was deleted, where $rank(m_i, w_j) = p$ and

1.	$x_{i,1} > 0$	$(1 \leq i \leq n)$
2.	$(x_{i,p} \neq 2 \wedge x_{i,p} > 0) \Rightarrow x_{i,p+1} > 0$	$(1 \leq i \leq n, 1 \leq p \leq l_i^m - 1)$
3.	$y_{j,q} \neq 2 \Rightarrow y_{j,q+1} \neq 2$	$(1 \leq j \leq n, 1 \leq q \leq l_j^w - 1)$
4.	$x_{i,p} > 0 \Rightarrow y_{j,q+1} \neq 2$	$(1 \leq i \leq n, 1 \leq p \leq l_i^m)$
5.	$y_{j,q} \neq 2 \Rightarrow x_{i,p} \neq 2$	$(1 \leq j \leq n, 1 \leq q \leq l_j^w)$
6.	$y_{j,1} > 0$	$(1 \leq j \leq n)$
7.	$(y_{j,q} \neq 3 \wedge y_{j,q} > 0) \Rightarrow y_{j,q+1} > 0$	$(1 \leq j \leq n, 1 \leq q \leq l_j^w - 1)$
8.	$x_{i,p} \neq 3 \Rightarrow x_{i,p+1} \neq 3$	$(1 \leq i \leq n, 1 \leq p \leq l_i^m - 1)$
9.	$y_{j,q} > 0 \Rightarrow x_{i,p+1} \neq 3$	$(1 \leq j \leq n, 1 \leq q \leq l_j^w)$
10.	$x_{i,p} \neq 3 \Rightarrow y_{j,q} \neq 3$	$(1 \leq i \leq n, 1 \leq p \leq l_i^m)$

Figure 5: The constraints for the 4-valued encoding of an instance SMI.

$rank(w_j, m_i) = q$. Assuming $q + 1 \leq l_j^w$, we consider the two cases separately.
Case (i) If m_i proposed to w_j during the execution of the MEGS algorithm, then w_j deletes all those men ranked below m_i on her preference list, so that in particular, $y_{j,q+1} \neq 2$.
Case (ii) If the pair (m_i, w_j) was deleted during the execution of the MEGS algorithm, then w_j must have received a proposal from a man m_k whom she prefers to m_i. Consequently, all men ranked below m_k on w_j's list are deleted by the MEGS algorithm, so that in particular, $y_{j,q+1} \neq 2$.

Now suppose that $y_{j,q} \neq 2$. Then by construction of the GS-domains, the MEGS algorithm deleted the man m_i such that $rank(w_j, m_i) = q$. So in addition, 2 is removed from the domain of $x_{i,p}$, where $rank(m_i, w_j) = p$, satisfying Constraint 5. Also, as in Case (ii) above, $y_{j,q+1} \neq 2$, satisfying Constraint 3.

Now consider Constraint 2 and suppose that $x_{i,p} \neq 2$ and $x_{i,p} > 0$. Then w_j has been removed from the list of m_i, where $rank(m_i, w_j) = p$. Also $x_{i,p} > 0$ implies that either (i) $p = 1$, or (ii) $x_{i,r} \neq 2$ $(1 \leq r < p)$. We consider the two cases separately.
Case (i) If $p = 1$, we have $x_{i,1} \neq 2$, and hence $x_{i,2} > 0$ by construction of the GS-domains.
Case (ii) As $x_{i,p} > 0$, it follows that $x_{i,r} \neq 2$ $(1 \leq r < p)$. Also $x_{i,p} \neq 2$. Hence $x_{i,r} \neq 2$ $(1 \leq r \leq p)$, so that $x_{i,p+1} > 0$ by construction of the GS-domains.

A similar argument can be used to verify that Constraints 6-10 are satisfied. Here the roles of the men and women are reversed and MEGS is replaced by WEGS. □

The two lemmas above, together with the fact that AC algorithms find the unique maximal set of arc consistent domains, lead to the following theorem.

Theorem 8. *Let I be an instance of SMI, and let J be a CSP instance obtained by the 4-valued encoding. Then the domains remaining after AC propagation in J correspond to the GS-lists of I in the following sense: for any i, j $(1 \leq i, j \leq n)$, $w_j \in GS(m_i)$ if and only if $\{2, 3\} \subseteq dom(x_{i,p})$, and similarly $m_i \in GS(w_j)$ if and only if $\{2, 3\} \subseteq dom(y_{j,q})$, where $rank(m_i, w_j) = p$ and $rank(w_j, m_i) = q$.*

In general AC may be established in $O(ed^r)$ time, where e is the number of constraints, d the domain size, and r the arity of each constraint [Bessière and Régin, 1997]. In the context of the 4-valued encoding, it follows that $e = O(L)$, $d = 4$ and $r = 2$, and hence AC may be enforced in time $O(L) = O(n^2)$. The time complexity of $O(L)$ is linear in the size of I and gives an improvement over the encoding presented in Section 2.1. Moreover $O(L)$ is also the time complexity of the EGS algorithm, which is known to be optimal [Ng and Hirschberg, 1990]. The space complexity of the 4-valued encoding is also $O(L)$.

Theorems 8 and 1(iii) show that we can find a solution to the CSP giving the man-optimal stable matching M_0 without search: for each man $m_i \in \mathcal{M}$, if $\{2, 3\} \not\subseteq dom(x_{i,r})$ for each r $(1 \leq r \leq l_i^m)$ then m_i is unmatched in M_0, otherwise we let p be the unique integer such that $dom(x_{i,p}) = \{1, 2, 3\}$ and define the partner of m_i to be the woman $w_j \in \mathcal{W}$ such that $rank(m_i, w_j) = p$. Considering the y_j variables in a similar way gives the woman-optimal stable matching M_z.

As in Section 2, we may go further and show that the CSP encoding yields all stable matchings in I without having to backtrack due to failure. As before we enumerate all solutions of I in a failure-free manner using AC propagation in J combined with a value-ordering heuristic, however in this case, maintenance of AC is much less expensive. The following theorem, proved in [Manlove and O'Malley, 2005], describes the enumeration strategy in this context.

Theorem 9. *Let I be an instance of SMI and let J be a CSP instance obtained from I using the 4-valued encoding. Then the following search process enumerates all solutions in I without repetition and without ever failing due to an inconsistency:*

– *AC is established as a preprocessing step, and after each branching decision, including the decision to remove a value from a domain;*

– *if all domains are arc consistent and some variable $x_{i,r}$ has $\{0, 1, 2, 3\}$ in its domain, then we let p be the unique integer such that $dom(x_{i,p}) = \{1, 2, 3\}$ and we choose p' to be the minimum integer $(p < p')$ such that $dom(x_{i,p'}) = \{0, 1, 2, 3\}$;*

– *the search proceeds by removing the value 3 from the domain of $x_{i,p'}$. On backtracking, the value 2 is removed from the domain of $y_{j,q}$, where $rank(m_i, w_j) = p$ and $rank(w_j, m_i) = q$;*

– *when a solution is found, it is reported and backtracking is forced.*

4 Concluding remarks

In this paper we have described two models for the Stable Marriage problem and its variant SMI as a CSP. Our first encoding is very natural and may be used to derive the GS-lists following AC propagation, although the time complexity for establishing AC is worse than that of the EGS algorithm. Our second encoding, whilst more complex, again yields the GS-lists, but this time the time complexity for AC propagation is optimal. Using both encodings we are able to find all stable matchings for a given instance of SMI using a failure-free enumeration without search.

A natural extension of this work is to the case where there is indifference in the preference lists. It has already been demonstrated [Gent and Prosser, 2002a; 2002b] that the earlier encodings of [Gent *et al.*, 2001] can be extended to the case where preference lists in a given SMI instance may include ties, suggesting that the same should be possible with the models that we present here. Another direction is to consider the Hospitals / Residents problem (HR) (a many-one generalisation of SMI). The $(n + 1)$-valued encoding from this paper, and the specialised constraints from [Unsworth and Prosser, 2005a; 2005b], have already been generalised to the HR case (see [Manlove *et al.*, 2005] for further details).

Finally, it remains to conduct an empirical investigation of the encodings presented in this paper, based on randomly-generated and real-world data. Such investigations have already been carried out for other encodings for SM and its variants [Gent and Prosser, 2002a; 2002b; Unsworth and Prosser, 2005a; 2005b].

Acknowledgements

We would like to thank Rob Irving and Patrick Prosser for helpful comments on earlier drafts of this paper. We would also like to thank Patrick Prosser for suggesting that previous (man-oriented and woman-oriented) versions of the models that we present here could be amalgamated.

References

[Aldershof and Carducci, 1999] B. Aldershof and O.M. Carducci. Refined inequalities for stable marriage. *Constraints*, 4:281–292, 1999.

[Bessière and Régin, 1997] C. Bessière and J-C. Régin. Arc consistency for general constraint networks: Preliminary results. In *Proceedings of IJCAI '97*, volume 1, pages 398–404. Morgan Kaufmann, 1997.

[Dye, 2001] J. Dye. A constraint logic programming approach to the stable marriage problem and its application to student-project allocation. BSc Honours project report, University of York, Department of Computer Science, 2001.

[Gale and Shapley, 1962] D. Gale and L.S. Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 69:9–15, 1962.

[Gale and Sotomayor, 1985] D. Gale and M. Sotomayor. Some remarks on the stable matching problem. *Discrete Applied Mathematics*, 11:223–232, 1985.

[Gent and Prosser, 2002a] I.P. Gent and P. Prosser. An empirical study of the stable marriage problem with ties and incomplete lists. In *Proceedings of ECAI '02*, pages 141–145. IOS Press, 2002.

[Gent and Prosser, 2002b] I.P. Gent and P. Prosser. SAT encodings of the stable marriage problem with ties and incomplete lists. In *Proceedings of SAT '02*, 2002. `http://gauss.ececs.uc.edu/Conferences/SAT2002/Abstracts/gent.ps`.

[Gent *et al.*, 2001] I.P. Gent, R.W. Irving, D.F. Manlove, P. Prosser, and B.M. Smith. A Constraint Programming Approach to the Stable Marriage Problem. In *Proceedings of CP '01*, volume 2239 of *Lecture Notes in Computer Science*, pages 225–239. Springer-Verlag, 2001.

[Green and Cohen, 2003] M.J. Green and D.A. Cohen. Tractability by approximating constraint languages. In *Proceedings of CP '03*, volume 2833 of *Lecture Notes in Computer Science*, pages 392–406. Springer-Verlag, 2003.

[Gusfield and Irving, 1989] D. Gusfield and R.W. Irving. *The Stable Marriage Problem: Structure and Algorithms*. MIT Press, 1989.

[Kato, 1993] A. Kato. Complexity of the sex-equal stable marriage problem. *Japan Journal of Industrial and Applied Mathematics*, 10:1–19, 1993.

[Knuth, 1976] D.E. Knuth. *Mariages Stables*. Les Presses de L'Université de Montréal, 1976. English translation in *Stable Marriage and its Relation to Other Combinatorial Problems*, volume 10 of CRM Proceedings and Lecture Notes, American Mathematical Society, 1997.

[Lustig and Puget, 2001] I.J. Lustig and J. Puget. Program does not equal program: constraint programming and its relationship to mathematical programming. *Interfaces*, 31:29–53, 2001.

[Manlove and O'Malley, 2005] D.F. Manlove and G. O'Malley. Modelling and solving the stable marriage problem using constraint programming. Technical Report TR-2005-192, University of Glasgow, Department of Computing Science, 2005.

[Manlove *et al.*, 2002] D.F. Manlove, R.W. Irving, K. Iwama, S. Miyazaki, and Y. Morita. Hard variants of stable marriage. *Theoretical Computer Science*, 276(1-2):261–279, 2002.

[Manlove *et al.*, 2005] D.F. Manlove, G. O'Malley, P. Prosser, and C. Unsworth. A Constraint Programming Approach to the Hospitals / Residents Problem. Technical Report TR-2005-196, University of Glasgow, Department of Computing Science, 2005.

[Ng and Hirschberg, 1990] C. Ng and D.S. Hirschberg. Lower bounds for the stable marriage problem and its variants. *SIAM Journal on Computing*, 19:71–77, 1990.

[Ng and Hirschberg, 1991] C. Ng and D.S. Hirschberg. Three-dimensional stable matching problems. *SIAM Journal on Discrete Mathematics*, 4:245–252, 1991.

[Ronn, 1990] E. Ronn. NP-complete stable matching problems. *Journal of Algorithms*, 11:285–304, 1990.

[Roth and Sotomayor, 1990] A.E. Roth and M.A.O. Sotomayor. *Two-sided matching: a study in game-theoretic modeling and analysis*, volume 18 of *Econometric Society Monographs*. Cambridge University Press, 1990.

[Roth, 1984] A.E. Roth. The evolution of the labor market for medical interns and residents: a case study in game theory. *Journal of Political Economy*, 92(6):991–1016, 1984.

[Thorn, 2003] M. Thorn. A constraint programming approach to the student-project allocation problem. BSc Honours project report, University of York, Department of Computer Science, 2003.

[Unsworth and Prosser, 2005a] C. Unsworth and P. Prosser. An n-ary constraint for the stable marriage problem. To appear in *Proceedings of the Fifth Workshop on Modelling and Solving Problems with Constraints*, 2005.

[Unsworth and Prosser, 2005b] C. Unsworth and P. Prosser. A specialised binary constraint for the stable marriage problem. To appear in *Proceedings of SARA '05, Lecture Notes in Computer Science*. Springer-Verlag, 2005.

[van Hentenryck *et al.*, 1992] P. van Hentenryck, Y. Deville, and C-M. Teng. A generic arc-consistency algorithm and its specializations. *Artificial Intelligence*, 57:291–321, 1992.

[Vate, 1989] J.E. Vande Vate. Linear programming brings marital bliss. *Operations Research Letters*, 8(3):147–153, 1989.

Modelling and Dynamic Symmetry Breaking in Constraint Programming

Karen E. Petrie
Cork Constraint Computation Center
University College Cork
Cork, Ireland
k.petrie@4c.ucc.ie

Abstract

Symmetry in constraint satisfaction problems can give rise to redundant search. The aim in symmetry breaking is to avoid such redundancy by excluding all but one example of each equivalence class of solutions. Two methods that have been developed to do this dynamically are Symmetry Breaking During Search and Symmetry Breaking via Dominance Detection. Modelling in CP means to move from a natural language specification of a problem, to a CSP formulation. This paper presents two case studies on the interaction between dynamic symmetry breaking and modelling.

1 Introduction

Combinatorial search is arguably the most fundamental aspect of Artificial Intelligence (AI) [2]. It is an extremely active research area, and has become very important commercially, through Constraint Programming (CP). Software packages such as ECL^{i}PS^e from IC-Parc [4] and ILOG Solver [17] are widely used on problems such as work force management at BT, resulting in savings of many millions for the companies concerned.

A Constraint Satisfaction Problem (CSP) consists of a set of variables each of which has a domain of values, and a set of constraints on the variables and values: a solution is an allocation of values to variables consistent with the constraints. A constraint solver *searches* for this solution by alternating phases of *branching* and *inference* to find an assignment of values to a set of variables which satisfies the constraints. The branching phase selects a variable and a possible value for it and seeks a solution in which it has that value. If no solution is found, then another value is tried. Branching thus causes the system to explore a tree of possible partial assignments, seeking one that can be completed. In the Inference phase, the solver attempts to deduce consequences of the constraint and the current partial assignment.

Modelling in CP means to move from a natural language specification of a problem, into a CSP instance consisting only of variables, values and constraints. It may be possible to find more than one model of a problem, in which case a model is sought that can efficiently lead to a solution through CSP solving techniques. This is where variable and value ordering heuristics fit into modelling process. This paper concentrates on the interaction of modelling and search with symmetry.

Constraint Satisfaction Problems (CSPs) are often highly symmetric. Symmetries may be inherent in the problem, as in placing queens on a chess board that may be rotated and reflected. Additionally the modelling of a real problem as a CSP can introduce extra symmetry: problem entities which are indistinguishable may in the CSP be represented by separate variables leading to $n!$ symmetries between n variables.

> **Definition of Symmetry** *Given a CSP L, with a set of constraints C, a symmetry of L is a bijective function f which maps a representation of a search state α to another search state, so that the following holds:*
>
> 1. *If α satisfies the constraints C, then so does $f(\alpha)$.*
> 2. *Similarly, if α is a no-good, then so too is $f(\alpha)$. [18]*

Symmetries can give rise to redundant search, while searching for solutions a partial assignments may be considered which is symmetric to one previously examined. If a partial assignment does not lead to a solution, neither will any symmetric assignment, and if it does lead to a solution, the new solution is symmetrically equivalent to one already found. To avoid this redundant search constraint programmers try to exclude all but one in each equivalence class of solutions. Many methods have been developed for this purpose. These symmetry exclusion methods can be divided into two classes: *static* and *dynamic*. Static symmetry breaking methods operate before search commences, and dynamic symmetry breaking methods operate during search.

In some classes of problems, the symmetry can be removed by remodelling the problem. For example, the golfers problem is: *32 golfers want to play in 8 groups of 4 each week, in such a way that any two golfers play in the same group at most once. How many weeks can they do this for?* This problem is highly symmetric. A possible model for this problem decides which group each player is assigned to in each week: the groups and the weeks (as well as the players) can be interchanged. By remodelling this problem using set variables, much of the symmetry can be removed [21].

Another static symmetry breaking method, involves adding constraints to the basic model. For instance, many problems (including the golfers problem above), have symmetry due to indistinguishable variables. Often, this symmetry can be removed by adding constraints that the value of these variables must be in ascending order. Crawford, Ginsberg, Lux and Roy developed a technique for constructing symmetry breaking ordering constraints for more general symmetries. It involves listing all possible permutations for each symmetry, then creating appropriate ordering constraints which allow only the first permutation to remain [5]. This technique effects the CP model both by the addition of constraints, and by fixing the variable ordering to be used during search.

In more recent years, Flener *et. al.* have concentrated on symmetry constraints for *matrix models*; where "a matrix model is a constraint program that contains one or more matrices of decision variables" [7]. For example the golfers problem can be modelled as a 3-d boolean matrix whose dimensions correspond to weeks, players and groups. A variable $x_{ijk} = 1$ iff in week i, player j plays in group k [21]. The orderings constraints which are proposed deal with *row* and *column* symmetries, where a *row(column)* symmetry of a 2-d matrix is a bijection between the variables of two of its rows(columns) that preserve solutions and non-solutions. Two rows(columns) are *indistinguishable* if their variables are pairwise indistinguishable due to a row (column) symmetry. A matrix model has *row(column) symmetry* iff all the rows(columns) of one of its matrices are indistinguishable. In the above matrix model of the golfers problem, the groups, weeks and the players are all indistinguishable, this results in row(column) symmetries.

In contrast to static symmetry breaking methods, dynamic symmetry breaking methods operate during the search process. The two dynamic symmetry breaking we will concentrate on in this paper are, symmetry breaking during search [1; 13], and symmetry breaking via dominance detection [6; 8]. More recently, computational group theoretic versions of these methods have been devised, namely GAP-SBDS [12] and GAP-SBDD [14].

Symmetry breaking during search (SBDS), was developed by Gent and Smith [13], having been introduced by Backofen and Will [1]. The search tree is built from decision points, where a decision point has two possible choices; either assign a value to a variable, or do not assign that value to that variable. When a decision point is first reached during search a value is assigned to a variable; if at a later stage in search the decision point is revisited then a constraint is imposed that the variable should not have the previously assigned value. SBDS operates by taking a list of symmetry functions (provided by the user) and placing related constraints when backtracking to a decision point and taking the second branch.

A feature of SBDS is that it only breaks symmetries which are not already broken in the current partial assignment: this avoids placing unnecessary constraints. A symmetry is broken when the symmetric equivalent of the current partial assignment is not consistent with that assignment. The following expression explains how SBDS works:

$$A \;\&\; g(A) \;\&\; var \neq val \;\Rightarrow g(var \neq val)$$

where A is the partial assignment made so far during search, $g(A)$ is the symmetric equivalent of A and $g(var \neq val)$ is the symmetrical equivalent to this failed assignment. If A is the current partial assignment and it has been established that $var \neq val$, it needs to be ensured that an unbroken symmetry is being dealt with, so a check is undertaken that $g(A)$ still holds. Then to ensure that the symmetrically equivalent subtree to the current subtree will not be explored, the constraint $g(var \neq val)$ is placed. An SBDS library is now available in the ECL^{i}PS^e constraint programming system [4]. As previously mentioned, SBDS requires a function for each symmetry in the problem describing its effect on the assignment of a value to a variable. If these symmetry functions are correct and complete, all the symmetry will be broken; as a result of this only non-isomorphic solutions will be produced. Although SBDS has been successfully used with a few thousand symmetry functions, many problems have too many symmetries to allow a separate function for each.

To allow SBDS to be used in situations where there are too many symmetries to allow a function to be created for each, Gent *et. al.* [12] have linked SBDS in ECL^{i}PS^e with GAP (Groups, Algorithms and Programming) [10], a system for computational algebra and in particular *computational group theory* (CGT). Group theory is the mathematical study of symmetry. GAP-SBDS allows the symmetry group to be specified compactly, using a set of generators, or for 'standard' groups by a canonical name. This avoids specifying individual group elements, which is not practical for large groups. GAP is used when a value is assigned to a variable, at a decision point, to find the *stabiliser* of the current partial assignment, i.e. the subgroup which leaves it unchanged. Then if the decision point is revisited on backtracking, the constraints are dynamically calculated from the stabiliser and placed accordingly. GAP-SBDS allows the symmetry to be handled more efficiently than in SBDS; the elements of the group are not explicitly created which is akin to what the symmetry functions represent in SBDS. However, there is an overhead in communication necessitated between GAP and ECL^{i}PS^e.

Symmetry Breaking via Dominance Detection (SBDD) [6; 8] performs a check at every node in the search tree to see if it is dominated by a symmetrically equivalent subtree already explored, and if so prunes this branch. In SBDD, the dominance detection function is based on the problem symmetry and is hard-coded for each problem. This means in practice SBDD can be difficult to implement, as the design of the dominance detection function may be complicated; the user has to ensure that all the symmetry of the problem is incorporated within the function to enforce full symmetry breaking.

Gent *et. al.* [14] have recently developed GAP-SBDD, a generic version of SBDD that uses the symmetry group of each problem rather than an individual dominance detection function and links SBDD (in ECL^{i}PS^e) with GAP. At each node in the search tree, ECL^{i}PS^e communicates the details of that node to GAP, and GAP returns false if dominance has been detected and that branch can be pruned, or true otherwise. Occasionally full dominance is not detected but there are variable/value pairs which are easily detected as being eligible for domain deletion; at which point GAP returns true followed by a list of variable/value pairs for which this is the case. ECL^{i}PS^e removes these values from the corresponding variables domains before search continues.

It is clear that static symmetry breaking methods effect the choice of model for a CSP. This situation is less clear for dynamic symmetry breaking methods. In general, dynamic symmetry breaking methods do not fix the CSP model, the only proviso is that the symmetry should be definable in terms of the search variables. This paper presents two cases studies which show how dynamic symmetry breaking and modelling techniques can interact. The first study shows that by considering both the model of the problem and the chosen symmetry breaking method an efficient method can be derived. The second study shows how the model chosen for a given problem can affect the choice of most efficient dynamic symmetry breaking method.

2 Case Study: SBDS and 'Peaceable Armies of Queens'

Robert Bosch introduced the "Peaceably Coexisting Armies of Queens" problem in his column in Optima in 1999 [3]. It is a variant of a class of problems requiring pieces to be placed on a chessboard, with requirements on the number of squares that they attack: Martin Gardner [11] discusses more examples of this class. In the "Armies of Queens" problem, we are required to place two equal-sized armies of black and white queens on a chessboard so that the white queens do not attack the black queens (and necessarily v.v.) and to find the maximum size of two such armies. Bosch asked for an integer programming formulation of the problem and how many optimal solutions there would be for a standard 8×8 chessboard.

A straightforward model of the problem has a variable s_{ij} to represent a square on row i, column j of the board:

$$\begin{aligned} s_{ij} &= 1 \text{ if there is a white queen on square } (i,j) \\ &= 2 \text{ if there is a black queen on square } (i,j) \\ &= 0 \text{ otherwise} \end{aligned}$$

If M is the region that is attached by a given square than, we can express the 'non-attacking' constraints as:

$$\begin{aligned} &s_{i_1j_1} = 1 \Rightarrow s_{i_2j_2} \neq 2 \\ \text{and } &s_{i_1j_1} = 2 \Rightarrow s_{i_2j_2} \neq 1 \text{ for all } ((i_1,j_1),(i_2,j_2)) \in M \end{aligned}$$

or more compactly as:

$$s_{i_1j_1} + s_{i_2j_2} \neq 3 \text{ for all } ((i_1,j_1),(i_2,j_2)) \in M$$

Tests in ECL^{i}PS^e show that, the single constraint gives the same number of backtracks as the two implication constraints, but is faster.

Constrained variables w, b count the number of white and black queens respectively (using the counting constraint: occurrences, provided in ECL^{i}PS^e). The last constraint is $w = b$, and the objective is to maximise w. This is achieved by adding a lower bound on w whenever a solution is found, so that future solutions must have a larger value of w; when there are no more solutions, the last one found has been proved optimal.

The model has n^2 search variables and approximately $4n^3$ binary constraints, as well as the counting constraints which have arity n^2.

Table 1 gives results for finding the optimal number of queens and proving that it is optimal, as well as for finding all optimal solutions. These experiments were run with a simple static variable ordering heuristic which searches the board: top row, left to right, then second row, left to right, and so on. The value ordering heuristic is the standard ECL^{i}PS^e one, which assigns values in numerical order starting with the smallest. The result for finding all solutions when $n = 8$ are missing as this result was not obtainable within the cut-off imposed of 1 hour.

2.1 SBDS in 'Armies of Queens'

The 'Armies of Queens' problem has the usual symmetry of the chessboard (reflection in the horizontal, vertical and both diagonal axes, and rotations through 90°, 180° and 270° and the identity); in addition, in any solution we can swap all the white queens for all the black queens, and we can combine these two kinds of symmetry. Hence the problem has 16 symmetries. SBDS is ideal for problems such as this since it only requires a simple function to describe the effect of each symmetry (other than the identity) on the assignment of a value to a variable. Hence, in this case, just 15 such functions are required.

The seven chessboard symmetry functions are labelled x, y, $d1$, $d2$, $r90$, $r180$, $r270$. The function which interchanges black and white is labelled BW; and the functions which combine the chessboard symmetries with interchanging black and white, are labelled as the board symmetries prefixed with BW. The symmetry functions takes a variable, s_{ij} and a possible value for this variable, v before returning the symmetric variable and the symmetric value as:

$$\begin{aligned}
x: &\quad s_{ij}, v \rightarrow \quad s_{i,n+1-j}, v \\
y: &\quad s_{ij}, v \rightarrow \quad s_{n+1-i,j}, v \\
d1: &\quad s_{ij}, v \rightarrow \quad s_{j,i}, v \\
d2: &\quad s_{ij}, v \rightarrow \quad s_{n+1-j,n+1-i}, v \\
r90: &\quad s_{ij}, v \rightarrow \quad s_{j,n+1-i}, v \\
r180: &\quad s_{ij}, v \rightarrow \quad s_{n+1-i,n+1-j}, v \\
r270: &\quad s_{ij}, v \rightarrow \quad s_{n+1-j,i}, v \\
bw: &\quad s_{ij}, v \rightarrow \quad s_{i,j}, [if\ v = 0\ then\ 0\ else\ 3 - v] \\
bwx: &\quad s_{ij}, v \rightarrow \quad s_{i,n+1-j}, [if\ v = 0\ then\ 0\ else\ 3 - v] \\
bwy: &\quad s_{ij}, v \rightarrow \quad s_{n+1-i,j}, [if\ v = 0\ then\ 0\ else\ 3 - v] \\
bwd1: &\quad s_{ij}, v \rightarrow \quad s_{j,i}, [if\ v = 0\ then\ 0\ else\ 3 - v] \\
bwd2: &\quad s_{ij}, v \rightarrow \quad s_{n+1-j,n+1-i}, [if\ v = 0\ then\ 0\ else\ 3 - v] \\
bwr90: &\quad s_{ij}, v \rightarrow \quad s_{j,n+1-i}, [if\ v = 0\ then\ 0\ else\ 3 - v] \\
bwr180: &\quad s_{ij}, v \rightarrow \quad s_{n+1-i,n+1-j}, [if\ v = 0\ then\ 0\ else\ 3 - v] \\
bwr270: &\quad s_{ij}, v \rightarrow \quad s_{n+1-j,i}, [if\ v = 0\ then\ 0\ else\ 3 - v]
\end{aligned}$$

Suppose that $n = 8$ and the first assignment places a white queen in the top left corner: $s_{1,1} = 1$. The symmetric assignments are: $x : s_{1,8} = 1$, $y : s_{8,1} = 1$, $d1 : s_{1,1} = 1$,

	Finding Optimal				Finding All Optimal Solutions		
n	No. of Backtracks to find first optimal solution	Total Number of Backtracks	Optimal Number of Queens	Time (secs)	Number of Backtracks	Number of Solutions	Time (secs)
2	0	1	0	0.0	1	1	0.0
3	1	2	1	0.0	17	16	0.0
4	4	28	2	0.01	149	112	0.02
5	190	265	4	0.16	383	18	0.20
6	1344	4998	5	3.63	9623	560	5.24
7	21882	93532	7	87.95	189013	304	132.99
8	802255	2716158	9	3215.2	-	-	-

Table 1: Results: Basic Model with no Symmetry Breaking

$d2 : s_{8,8} = 1$, $r90 : s_{1,8} = 1$, $r180 : s_{8,8} = 1$, $r270 : s_{8,1} = 1$, $bw : s_{1,1} = 2$, $bwx : s_{1,8} = 2$, $bwy : s_{8,1} = 2$, $bwd1 : s_{1,1} = 2$, $bwd2 : s_{8,8} = 2$, $bwr90 : s_{1,8} = 2$, $bwr180 : s_{8,8} = 2$, $bwr270 : s_{8,1} = 2$. All the symmetries which swap black and white, apart from bw are inconsistent with $s_{1,1} = 1$, because the symmetrically equivalent assignment would place a black queen in one of the corners where it could be attacked by the first assignment, so these symmetries are no longer considered on this branch. On backtracking to the first choice point, where $s_{1,1} = 1$ is set, and taking the alternative branch of $s_{1,1} \neq 1$, the symmetry functions are used to calculate the symmetric variables ($SymVar$) and values ($SymVal$). Lastly constraints of the form $SymVar \neq SymVal$ are placed in order to stop the subtree symmetric to this from ever being explored. This process ensures that if a white queen can not be placed in the top corner, then a queen is never placed in any of the corners. This can be seen in figure 2.1 where all the symmetry breaking constraints placed by SBDS are indicated by $\diamondsuit$.

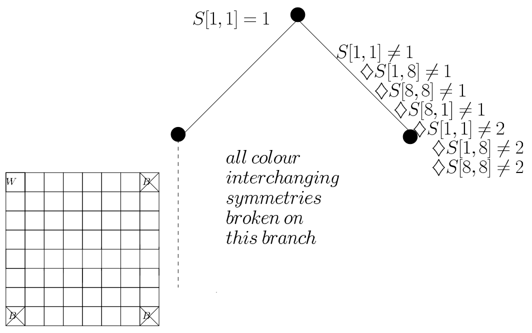

$S[1,1]=1$
$S[1,1]\neq 1$
$\diamondsuit S[1,8]\neq 1$
$\diamondsuit S[8,8]\neq 1$
$\diamondsuit S[8,1]\neq 1$
$\diamondsuit S[1,1]\neq 2$
$\diamondsuit S[1,8]\neq 2$
$\diamondsuit S[8,8]\neq 2$
all colour interchanging symmetries broken on this branch
W

Figure 1: SBDS operating on the armies of queens problem

Table 2 shows the empirical results when SBDS is integrated into the simple CP model outlined in section 2. Comparing this with Table 1 shows that SBDS gives a factor greater than 5 improvement in number of backtracks for the $n = 8$ case. However, the runtime increases when SBDS is used. This is because the first value chosen by the value ordering heuristic represents an empty square on the chessboard. The symmetry breaking constraints placed by SBDS when backtracking from these assignments, will forbid placing an empty square in a symmetrically equivalent position. These constraints occur an overhead and are not useful in steering search towards improved solutions. In fact as better, solutions with more queens on the board are found they become redundant. Later on in search, when leaving empty squares has been tried, values 1 then 2 will be allocated, which relate to placing white and black queens respectively. When SBDS is triggered through backtracking past failed cases of these assignments more useful constraints are returned. These constraints are the ones that operate to reduce the number of backtracks so significantly. In general, when trying to anticipate the effect of SBDS on a given model, it is worth considering the variable and value ordering heuristics. If these heuristics will lead to placing constraints early in search which, will have little effect at the time, then become vacuous at a later stage of search, it is worth considering if a better heuristic can be found.

2.2 Value Ordering and SBDS

The value ordering heuristic which places empty squares first can also hinder the optimisation process. The first solution to be found has 0 allocated to every square, which is equivalent to an empty board. This gives a lower bound of 0 for the maximum number of white queens which can be placed on the board. A constraint is then posted which says that the next number of white queens must be greater than this lower bound which in this case would be > 0. The process continues by increasing the lower bound in integer increments until the optimum number (m) is found. At this point, the program searches for a solution with maximum number of white queens $m + 1$; on failing to find one it has proven that m is indeed the optimum. If instead of allocating empty squares in the initial stages, queens are placed on squares first, the earliest solution found gives, a better lower bound for the optimum. In this case the program commences by placing as many white queens as possible then as many black queens as possible, only allocating empty squares when no queens can be placed. The lower bound then becomes the number of black or white queens (there is a constraint to ensure they are equal) on the board (p). Optimisation continues as before, by setting a constraint which states that the next value found must be greater than p. This value ordering heuristic is also potentially a good heuristic with respect to SBDS. The first decisions made relate to placing queens on the board, if these

	Finding Optimal				All Solutions		
n	No. of Backtracks to find first optimal solution	Total Number of Backtracks	Optimal Number of Queens	Time (secs)	Number of Backtracks	Number of Solutions	Time (secs)
2	0	1	0	0.1	1	1	0.0
3	1	2	1	0.03	2	1	0.03
4	4	9	2	0.10	16	10	0.10
5	68	70	4	0.60	64	3	0.52
6	462	886	5	7.30	1286	35	9.19
7	6994	15538	7	138.16	24106	19	181.310
8	298235	473141	9	4454.45	-	-	-

Table 2: Results: Basic Model with SBDS

decisions are backtracked past at a later stage, than SBDS can place constraints which state that a queen should not be placed in the given square. These constraints are useful in directing search. In optimisation problems, by considering the best heuristic for a problem through knowledge of the optimisation process, than a good heuristic for SBDS may also be derived, as the extra information given to the optimisation process can relate to SBDS placing more informative symmetry breaking constraints. In general, by considering the best heuristics for a given problem, a good heuristic will also be found with respect to SBDS, as the heuristic chosen will build a search tree which starts by trying the mostly likely value for a variable, this relates to the scope of constraints that SBDS can place.

It is possible to implement this new strategy as a value ordering heuristic which tries 1 before 2, before 0; hence it implements allocating queens to squares on the board before leaving them empty. However, this heuristic does have a time overhead as a decision process has to be undertaken at each search variable to see which value should be allocated. A less complex approach is to reassign the values so that $0 = white\ queen$, $1 = black\ queen$ and $2 = empty\ square$. Then allocate 0 before 1, before 2 as before. In SBDS this approach does necessitate a minor change to the symmetry functions which interchange black and white queens.

2.3 Variable Ordering and SBDS

In the previous experiments in section 2.1, a static variable ordering heuristic was used which assigned the top row of the board from left to right followed by the second row from left to right until all the variables were assigned. If constraints were being used to break the symmetry this static ordering may be mandatory, as often the variable order must be defined before search commences, in order to ensure these methods are complete and no solutions are lost [5]. If SBDS is the symmetry breaking method chosen, this information is not needed before search commences, so the use of dynamic variable ordering is permitted, and can be easily integrated with the SBDS library. A dynamic variable ordering chooses the next variable to be allocated during search, according to the search decisions and the resulting propagation to that point. A common and well proven heuristic is smallest domain first (SDF), which allocates the next variable to be assigned a value to be the one with the smallest number of entries in its domain.

2.4 Experimental Results of Combining Variable and Value Ordering Heuristics with SBDS

Table 3 Contains the results of combining the value ordering heuristic outlined in section 2.2 and SDF variable ordering as discussed in section 2.3 with SBDS.

Comparing the previous results for the basic model with SBDS shown in table 2 with the more advanced model results shown in table 3, shows a large reduction in time for all cases. For $N = 8$ the reduction in time is 10 fold to find the optimal number of 'Queens' that can be placed on the board, where N is the length of one side of the chess board. The reduction in the total number of backtracks for $N = 8$ is equally impressive at 10 fold again, but the most impressive reduction comes in the number of backtracks to find the first optimal solution which is reduced by a factor greater than 1000 for the $N = 8$ case. This means that good lower bounds for the optimum are being found early in search. In the $N = 8$ case it can be seen that the lower bound is 6 for the new value ordering, whereas it was 0 in the original case, the actual optimal value is 9 so 6 is a good approximation.

Turning to finding all the solutions to the problem it can be seen that there is a great reduction in both backtracks and time, between the original model and the current model. It is possible to prove that there are 71 non-isomorphic results for $N = 8$, with the approved variable and value ordering heuristic, this is a new result.

Looking back at the results without SBDS (table 1) it can be seen that these new results outperform those, both in terms of time and backtracks, in all cases. Studying the model with the variable and value ordering heuristics but with SBDS (not shown due to space constraints) it can be seen that the addition of SBDS offers a saving in both time and number of backtracks. This shows that the combination of modelling techniques and the SBDS library can be very powerful in efficiently solving problems.

3 Case Study: SBDS versus SBDD and 'Graceful Graphs'

There is limited past work comparing GAP-SBDS and GAP-SBDD. Harvey [15] studied the algorithms theoretically and

	Finding Optimal					All Solutions	
n	Lower bound on optimum	Optimum No. of Queens	No. of Bt. to find first optimal solution	Total Number of Bt.	Time (secs)	Number of Bt.	Time (secs)
2	0	0	0	1	0.0	1	0.0
3	1	1	0	2	0.02	2	0.0
4	2	2	0	4	0.05	12	0.05
5	3	4	3	12	0.17	23	0.22
6	4	5	1	153	1.79	405	3.16
7	5	7	9	2231	21.23	5186	47.90
8	6	6	266	46894	406.39	106940	752.11

Table 3: Results: SBDS with SDF Variable Ordering & Value Ordering heuristic

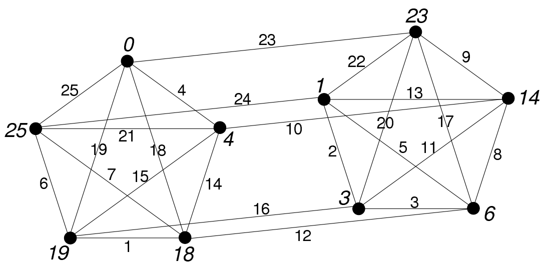


Figure 2: Graceful labelings of $K_5 \times P_2$ and the Double Wheel DW_5

concluded that SBDS and SBDD are closely related, the difference being where in the search tree, and how, symmetry breaking is enforced. Gent *et al.* [14] applied GAP-SBDS and GAP-SBDD to instances of the balanced incomplete block design (BIBD) problem and showed that GAP-SBDD could solve much larger problems, and was faster than GAP-SBDS on the smaller problems which both could solve. They surmised that this was due to the communication overhead between GAP and ECL^{i}PS^e, since the overhead in GAP-SBDD, which usually returns only a Boolean answer, is less than in GAP-SBDS, where a set of constraints is returned.

On the other hand, Petrie and Smith [19] found that in *Graceful Graphs* problems, GAP-SBDS outperformed GAP-SBDD on all instances studied. In the next section, the reason for this difference in performance is identified.

3.1 Graceful Graphs

A labeling f of the vertices of a graph with q edges is *graceful* if f assigns to each vertex a unique label from $\{0, 1, ..., q\}$ and, when each edge xy is labeled with $|f(x) - f(y)|$, the edge labels are all different [9]. (Hence, the edge labels are a permutation of $1, 2, ..., q$.) Figure 3.1 shows an example.

Lustig and Puget [16] give a constraint model for finding a graceful labeling of a graph. A basic CSP model has a variable for each node $x_1, x_2, ..., x_n$, each with domain $\{0, 1, ..., q\}$ and a variable for each edge $d_1, d_2, ..., d_q$, each with domain $\{1, 2, ..., q\}$. The constraints of the problem are: if edge k joins nodes i and j then $d_k = |x_i - x_j|$; $x_1, x_2, ..., x_n$ are all different; and $d_1, d_2, ..., d_q$ are all different.

ECL^{i}PS^e provides two different levels of propagation for the *alldifferent* constraint. It can either be treated as a clique of binary $\neq$ constraints or as a *global alldifferent* which does more propagation. We use the *global alldifferent* on the edge variables and the binary $\neq$ version on the node variables. They are treated differently because the values assigned to the edge variables form a permutation and hence give more scope for domain pruning than the node variables, which have more possible values than variables. The node variables are used as the search variables. More information on the modeling of this problem and the symmetry group is given by [19].

The graph $K_5 \times P_2$, shown in Figure 3.1, consists of two copies of K_5, with corresponding vertices in the two cliques forming the vertices of a path P_2. The symmetries of $K_5 \times P_2$ are: first, any permutation of the 5-cliques which act on both in the same way. Second, inter-clique symmetry: all the node labels in the first clique can be interchanged with the labels of the adjacent nodes in the second. Third, complement symmetry: every vertex label x_i can be replaced by its complement $q - x_i$. The graph symmetries and the complement symmetry can be combined with each other. Hence, the size of the symmetry group is $5! \times 2 \times 2$. In general, $K_m \times P_2$ graphs have a symmetry group of size $m! \times 2 \times 2$. This study concentrates on symmetry breaking in 3 such graphs, with $m =$ 3, 4 and 5. The results of finding all graceful labelings of these graphs using either GAP-SBDS or GAP-SBDD can be found in Table 4. (All experiments in the paper were run on a 1.6GHz Pentium 4 processor with 512MB of memory, using ECL^{i}PS^e version 5.7 and GAP version 4.2.) From Table 4, it can be seen that GAP-SBDD is slower than GAP-SBDS for all instances. This is also true for other graphs, as shown by [20].

3.2 Analysis

To explain why GAP-SBDS is faster than GAP-SBDD for finding graceful labelings of graphs with symmetry, we have analysed the behaviour of GAP-SBDS and GAP-SBDD for the three graphs $K_3 \times P_2$, $K_4 \times P_2$ and $K_5 \times P_2$. The reasons for the differences in search are consistent, but for simplicity only the results for $K_3 \times P_2$ are presented here.

It should be noted that Table 4 gives the number of *deep backtracks*. We use the term deep backtrack when the search has progressed beyond a decision point, but then later has to revisit it. A *shallow backtrack* occurs when propagating the

		BT	ECL^{i}PS^e time	GAP time	Total time
GAP-SBDD	$K_3 \times P_2$	13	0.23	0.50	0.73
	$K_4 \times P_2$	173	7.18	2.72	9.90
	$K_5 \times P_2$	4402	337.69	88.20	426.89
GAP-SBDS	$K_3 \times P_2$	9	0.20	0.33	0.53
	$K_4 \times P_2$	165	7.15	1.35	8.50
	$K_5 \times P_2$	4390	352.10	36.61	388.71

Table 4: Comparison of GAP-SBDS and GAP-SBDD showing backtracks (bt) and the time (in seconds) for finding all graceful labelings of $K_3 \times P_2$, $K_4 \times P_2$, $K_5 \times P_2$.

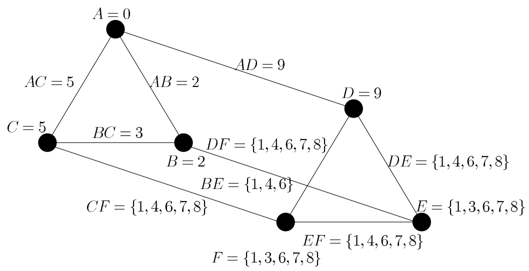


Figure 3: The domains of the node and the edge variables after propagating $C = 5$, using GAP-SBDD

constraint $var = val$ on the left branch of a decision point causes a failure, and the $var \neq val$ branch is taken instead. Most constraint programming systems count the number of deep backtracks, but in this case it does not accurately reflect differences in search. In GAP-SBDS, symmetry-breaking constraints can be added whenever the left ($var \neq val$) branch is followed, including after a shallow backtrack.

Figure 4 shows the search trees created by GAP-SBDS and GAP-SBDD in finding all graceful labelings of $K_3 \times P_2$, from the point where the first difference occurs, which is after the first two solutions (from 4 in total) have been found. The variable names A to E in Figure 4 correspond to the nodes shown in Figure 3; the edges and corresponding edge variables are named by a pair of letters corresponding to the nodes defining the edge.

After assigning $C = 5$, GAP-SBDS immediately reverses from this decision to follow the $C \neq 5$ branch (a shallow backtrack), whereas GAP-SBDD continues, setting $E = 1$, before returning to take the $C \neq 5$ branch later in search (a deep backtrack).

The difference in the search trees is due to differences in constraint propagation. GAP-SBDD arrives at the search state shown in Figure 3. One of the edges must be labeled 9 (the number of edges in the graph) and the adjacent nodes must be labeled 0 and 9. At this stage $A = 0$ and B and C are labeled with values other than 9; hence D, the only other node adjacent to A, must take the value 9, and this inference is made by constraint propagation. Figure 3 shows the variable domains at this point. Because there are already edges labeled 2 (AB) and 3 (BC), the edges DE and DF cannot have those values, and hence E and F cannot have the values 6 or 7. Using GAP-SBDS, the domains of the variables are also reduced by symmetry-breaking constraints previously added on this branch. Those that are relevant in this case are symmetric equivalents of $B \neq 1$, namely $E \neq 1$, $F \neq 1$, $E \neq 8$ and $F \neq 8$. (Because of the graph symmetry, nodes E and F are symmetric to node B, and the value 8 is symmetric to the value 1 because of the complement symmetry.) The only remaining value in the domains of both E and F is 3, and since these variables must have different values, this branch fails.

Most of this propagation cannot occur in GAP-SBDD. GAP just returns a boolean to indicate whether the current node is dominated or not, and possibly a list of values to prune from the domains of specific search variables. In the current implementation, a variable/value pair is returned for domain pruning if its assignment would cause dominance to be detected. In this case E/1, F/1, E/8 and F/8 are not returned. Although GAP-SBDD successfully breaks the symmetry (in this case by detecting dominance when the assignment $E = 1$ is tried) posting SBDS constraints at an earlier stage can clearly lead to earlier pruning.

The reason this difference between GAP-SBDS and GAP-SBDD is highlighted by experimentation on this problem, as oppose to on the other problems consider by Gent *et al.* [14], relates directly to the model of the problem; Specifically to the fact that the search variables are not the most constrained variables with the model. GAP-SBDS breaks symmetry by placing constraints, these constraints can propagate with all the variables within the model. GAP-SBDD provides no information which could be related to the variables not directly involved in search.

4 Conclusion

Symmetry exclusion methods can be divided into two classes: static and dynamic. Static symmetry breaking methods operate before search commences, whereas dynamic symmetry breaking methods operate during search. Static symmetry breaking methods generally require a fixed model with static variable and value ordering heuristics. Dynamic symmetry breaking methods give the CP practitioner more freedom as to which model to chose; the only proviso is that it must be possible to define the symmetry in terms of the search variables.

In this paper, through the use of two case studies, we have shown how the CP model, can interact with dynamic symmetry breaking methods. The first case study illustrated how dynamic symmetry breaking and modelling can interact to provide an efficient method for solving a problem. The second case study shows how the model chosen for a given problem, can affect the choice of most efficient dynamic symmetry breaking method.

This paper represents a preliminary study, showing that, by combining modelling techniques and dynamic symmetry breaking, more efficient solving techniques can be derived than by considering either of these aspects individually. Further work in this area is needed if the exact relationship between symmetry breaking and modelling is to be fully understood.

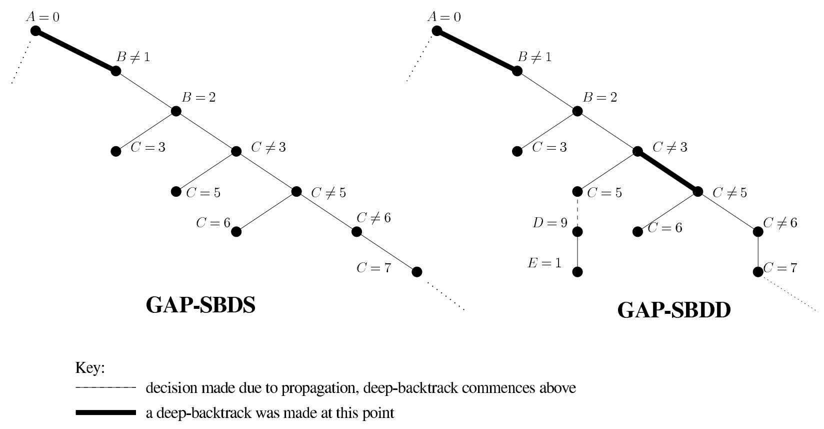

$A=0$
$B\neq 1$
$B=2$
$C=3$
$C\neq 3$
$C=5$
$C\neq 5$
$C=6$
$C\neq 6$
$C=7$
$D=9$
$E=1$
GAP-SBDS
GAP-SBDD
Key:
decision made due to propagation, deep-backtrack commences above
a deep-backtrack was made at this point

Figure 4: The search tree branch where GAP-SBDS and GAP-SBDD differ

References

[1] R. Backofen and S.Will. Excluding symmetries in constraint-based search. In Joxan Jaffar, editor, *Proc. of CP'99*, LNCS 1713, pages 73–87. Springer, 1999.

[2] Roman Bartak. Guide to Constraint Programming. Technical report, Charles University Prague, 1998.

[3] Robert A. Bosch. Peaceably coexisting armies of queens. *Optima (Newsletter of the Mathematical Programming Society)*, 62:6–9, 1999.

[4] A. M. Cheadle, W. Harvey, A. J. Sadler, J. Schimpf, K. Shen, and M. G. Wallace. ECLiPSe: An introduction. Technical Report IC-Parc-03-1, IC-Parc, 2003. `www.icparc.ic.ac.uk/eclipse/` .

[5] James Crawford, Matthew L. Ginsberg, Eugene Luki, and Amitabha Roy. Symmetry-breaking predicates for search problems. In Luigia Carlucci Aiello, Jon Doyle, and Stuart Shapiro, editors, *KR'96: Principles of Knowledge Representation and Reasoning*, pages 148–159. Morgan Kaufmann, San Francisco, California, 1996.

[6] Torsten Fahle, Stefan Schamberger, and Meinolf Sellmann. Symmetry breaking. In Toby Walsh, editor, *Proc. of CP'01*, LNCS 2239, pages 93–107. Springer, 2001.

[7] P. Flener, A.M. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, and T. Walsh. Breaking row and column symmetries in matrix models. In P. Van Hentenryck, editor, *Proc. of CP'02*, LNCS 2470, pages 462–476. Springer, 2002.

[8] Filippo Focacci and Michela Milano. Global cut framework for removing symmetries. In Toby Walsh, editor, *Proc of CP'01*, LNCS 2239, pages 77–92. Springer, 2001.

[9] J.A. Gallian. A Dynamic Survey of Graceful Labeling. In *The Electronic Journal of Combinatronics*, 2002. `(http://www.combinatorics.org/Surveys)` .

[10] The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.2*, 2000. `(http://www.gap-system.org)` .

[11] Martin Gardner. Chess queens and maximum unattacked cells. *Math Horizon*, pages 12–16, November 1999.

[12] I. P. Gent, W. Harvey, and T. Kelsey. Groups and constraints: Symmetry breaking during search. In P. Van Hentenryck, editor, *Proc. of CP'02*, LNCS 2470, pages 415–430. Springer, 2002.

[13] I. P. Gent and B. M. Smith. Symmetry breaking in constraint programming. In *Proc. of ECAI-2002*, pages 599–603. IOS Press, 2000.

[14] Ian P. Gent, Warwick Harvey, Tom Kelsey, and Steve Linton. Generic SBDD Using Computational Group Theory. In Francesca Rossi, editor, *Proc. of CP'03*, LNCS 2833, pages 333–347. Springer, 2003.

[15] Warwick Harvey. Symmetry Breaking and the Social Golfer Problem. In *Proc. SymCon-01: Symmetry in Constraints*, pages 9–16, 2001.

[16] I.J.Lustig and J.-F. Puget. Program Does Not Equal Program: Constraint Programming and Its Relationship to Mathematical Programming. In *INTERFACES*, volume 31(6), pages 29–53, 2001.

[17] ILOG. *ILOG Solver 5.0 User's Manual*, 2001.

[18] I. McDonald and B. M. Smith. Partial symmetry breaking. In *Proc. of CP'02*, LNCS 2470, pages 431–445. Springer, 2002.

[19] K. E. Petrie and B. M. Smith. Symmetry breaking in graceful graphs. In *Proc. of CP'03*, LNCS 2833, pages 930–934. Springer, 2003.

[20] Karen Petrie. Why SBDD can be worse than SBDS. In *Proc. SymCon-03: Symmetry in Constraints*, pages 168–176, 2003.

[21] Barbara M. Smith. Reducing Symmetry in a Combinatorial Design Problem. Technical report, School of Computer Studies, University of Leeds, January 2001.

Exploring the use of constraint programming for enforcing connectivity during graph generation

Kenneth N. Brown[1], **Patrick Prosser**[2], **J. Christopher Beck**[3] **and Christine Wei Wu**[1]

[1]Cork Constraint Computation Centre, Department of Computer Science,
University College Cork, Ireland.
{k.brown,cww1}@cs.ucc.ie
[2]Department of Computer Science, University of Glasgow, Scotland
pat@dcs.gla.ac.uk
[3]Toronto Intelligent Decision Engineering Laboratory,
Department of Mechanical and Industrial Engineering, University of Toronto, Canada.
jcb@mie.utoronto.ca

Abstract

We discuss the problem of using constraint models to force generated graphs to be connected. We represent the graph as a simple adjacency matrix, and then attempt to post constraints ensuring connectivity. Doing this using standard modelling primitives is harder than expected, because of a problem with our use of the implication operator. We develop a global constraint *connected-graph*, and show that it does save time over a class of graph generation problems, but most of the gains come from simple pre-search filters applied to insoluble instances. We finish by discussing a new constraint, *graphical*, which simply ensures that a partially instantiated graph can be completed.

1 Introduction

We consider the problem of enforcing connectivity while generating graphs, a problem which appears embedded within many practical applications. For example, in computational chemistry, generating all possible single molecules formed from a set of atoms involves generating connected multigraphs (where atoms are represented by vertices, valencies are represented by the degrees, and bonds are represented by edges [Wu, 2004]). In telecommunications network planning, base stations and hubs are vertices, the communication links are edges, and the network must clearly be connected. From Operations Research, solving the Travelling Salesperson Problem involves constructing a minimal length path which visits every node: i.e. a connected graph where every vertex has degree 2. Many such problems come with associated side constraints - for example, legal bonds between atoms or acceptable delays on communications links - and thus a constraint programming solution may be desirable. In this paper, we attempt to develop a constraint model which ensures that generated graphs are connected. There has been some previous work on reasoning about properties of graphs using constraint models. For example, [Le Pape *et al.*, 2002] present a new variable type representing paths in a graph, [Sorlin and Solnon, 2004] discusses a global constraint for graph isomorphism problems, and [Pesant and Soriano, 2002] generate optimal cycle covers for networks. Graph theoretic algorithms have been used extensively in constraint programming; see [Régin and Gomes, 2004], for example, or [Simonis, 2004] for a survey.

We have investigated a pure version of the problem, constructing undirected connected simple graphs with no self loops over a set of vertices with known degrees. First, we represent the undirected simple graphs with no self loops. We assume that we are given an empty adjacency matrix and a degree sequence, and that the search process is free to select or reject any edge. Our goal is then to construct constraint-based models which will allow us to enforce connectivity while searching for all graphs that realise a given sequence.

We start by presenting a constraint encoding for producing the basic graphs with the specified degree sequence. We then consider how to enforce connectivity on these graphs using standard constraints, but run into problems. We then present *connected-graph*, a global constraint for maintaining connectivity, and describe a first implementation based on a 'connected-components' algorithm. We then extend that implementation to include some limited propagation. We present some results indicating that the constraint is effective when searching for all solutions to a set of degree sequences, but that most of the efficiency gains come from sequences which have no connected realisation. We then conclude by looking at a more limited constraint which simply enforces graphicality - that is, it ensures that a partially instantiated graph can in fact be fully realised as a graph.

2 Graph Theory Preliminaries

We assume the necessary constraint background, and concentrate here on introducing the graph theory terminology and basic graph-theoretic results. See [Gould, 1988] for a more comprehensive (and readable) account.

A *graph* G is a finite set $V = \{1, 2, \ldots, n\}$ of *vertices*, and a collection $E = \langle \{v_1, w_1\}, \ldots, \{v_m, w_m\} \rangle$ of *edges*, where each $v_i, w_i \in V$. If the edges in E are ordered pairs rather than sets, then E is a *directed* graph. If the list E is a set, then G is a *simple* graph; if E contains multiple copies of an edge, then G is a *multigraph*. If $\{v_i, v_i\} \in E$, then it is a *self-loop*. From now on, we will assume that E is an undirected simple graph with no self loops.

If $\{v_i, v_j\} \in E$, then v_i and v_j are *adjacent*, and the edge $\{v_i, v_j\}$ is *incident* on both v_i and v_j. Two vertices v_1 and v_k are *connected* if there is a *path* $P = \langle v_1, v_2 \ldots, v_k \rangle$ of vertices, such that $\forall i < k,\ \{v_i, v_{i+1}\} \in E$. A graph is *connected* if $\forall i, j,\ v_i$ and v_j are connected. A *connected component*, C_i, is a set of vertices such that $v_j \in C_i \leftrightarrow \forall v_k \in C_i, v_j$ and v_k are connected. A connected graph has exactly one connected component. A connected graph must have at least $n-1$ edges. A graph has at most $n * (n-1)/2$ edges.

The *degree* of a vertex is the number of edges incident on it. By the *handshaking lemma*, the sum of the degrees of a graph is twice the number of edges, and so, as a corollary, the sum of the degrees of the vertices must be even. Let $degree(i)$ be the degree of vertex i. Then $D = \langle degree(1), \ldots degree(n) \rangle$ is the *degree sequence* of G. We can now pose the question: given V (a set of n vertices), and D (a sequence of n integers), does there exist a graph G that *realises* D? That is, can we construct an edge set E such that D is the degree sequence of $G = (V, E)$? Further, can we ensure that G is connected, and can we generate all connected realisations?

3 Representing a simple graph with a given degree sequence

Representing a simple graph with a given degree sequence [Shiloach, 1981] is straightforward. For a problem with n vertices, we create an $n \times n$ array, A, of constrained 0/1 variables. When $A[i, j] == 1$, there is an edge from i to j, and when $A[i, j] == 0$, there is no edge from i to j. Since our graphs are undirected, we post the constraint $A[i, j] == A[j, i]$ for each pair $i < j$. To stop self-loops we add the constraint $A[i, i] == 0$ for each i. If we then assume that D is an array of integers, such that $D[i]$ is the degree of vertex i, then we can post a constraint for each row i of A to ensure that the vertices have the correct degree: $\sum_{j=1}^{n} A[i, j] == deg[i]$.

We can now ask the solver to generate all solutions. For a degree sequence $\langle 2, 2, 2, 1, 1 \rangle$, we will get 7 solutions: 6 of them will be paths, and the 7th will consist of two components, one with a pair of connected vertices (K_2), and the other a triangle (K_3). Obviously, the first 6 are isomorphic to one another, and the 7th is disconnected. For a degree sequence $\langle 2, 2, 1, 1, 1 \rangle$, there are no solutions, since the sum of the degree sequence is odd (from the handshaking lemma). Our model, however, requires search to discover this. Therefore there are two extensions required: we must detect choices which would lead to a disconnected graph, and need to identify degree sequences which cannot be realised.

4 Enforcing connectivity: a first attempt

Since connectivity is defined in terms of paths, we first considered enforcing connectivity by introducing n^2 *path* variables $P[i, j]$, such that $P[i, j] = 1$ if there is a path from vertex i to vertex j. The path variables are not intended to be decision variables, but will be linked to the adjacency matrix. Each time we set two vertices to be adjacent (i.e. set $A[i, j] = 1$), we also set $P[i, j] = 1$, and propagate recursively to other path variables. Therefore, we add constraints $A[i, j] == 1 \rightarrow P[i, j] == 1$ (for each pair i and j), and $(P[i, j] == 1 \wedge P[j, k] == 1) \rightarrow P[i, k] == 1$ (for all triples i, j and k). We then add a constraint forcing every vertex to be connected to vertex 1: $P[1, j] == 1$ for all j.

But this doesn't work, and it doesn't work because we have relied on implication. $P \rightarrow Q$ is true when Q is true and P is false, so our encoding allows a solver to cheat by setting $P[i, j] = 1$ whenever it needs to, and thus our path-connectivity constraint is trivially satisfied.

5 Enforcing connectivity: a second attempt

There are two standard algorithms for checking whether or not a graph is connected ([Cormen *et al.*, 2001]): depth-first search, and CONNECTED-COMPONENTS(G). This second algorithm maintains data structures for the connected subcomponents of the graph, and its outline is sketched below:

```
Connected-Components(G)
1. for each vertex v in V(G)
2.    MakeSet(v)
3. for each edge (v,u) in E(G)
4.    if FindSet(u) != FindSet(v)
5.       Union(u,v)
```

The algorithm starts by producing an individual set for each vertex in the graph, such that each set contains exactly one vertex. We then iterate over the edges of the graph, combining pairs of sets if they span an edge. On termination, the sets represent the components of the graph: if there is only one component, then the graph is connected, and otherwise it is disconnected. In line 2, MakeSet(v) creates a new set containing vertex v. In line 4, FindSet(u) returns the set that contains u, and in line 5, Union(u,v) unions the sets that contain u and v. Let n be the number of vertices, and e the number of edges. We assume that Union(u,v) takes $O(|v|)$ operations, and that FindSet and MakeSet take $O(1)$. For a connected graph, $n-1$ of the edges require an application of Union (to establish that each of the remaining vertices connects to the first). In the worst case, the algorithm always applies Union(u,v) when u is a singleton set, and $|v|$ steps from 1 to $n-1$, and thus requires $1+2+\ldots+(n-1) = (n-1)n/2$ operations, plus n operations for the initial sets. Thus the worst case running time is $O(n^2)$. The space required is $O(n)$ (for initially n singleton sets, and finally 1 set of n elements).

We attempted to construct a declarative constraint encoding of this algorithm using the set variables provided in Choco. We introduced n set variables $S[i]$, where $S[i]$ is initialised with the value $\{i\}$. Then, when search selects the edge (i, j), we want to combine sets $S[i]$ and $S[j]$. So we added the following constraint:

$$\forall i\, \forall j\ A[i, j] = 1 \rightarrow ((S[i] \subseteq S[j]) \wedge (S[j] \subseteq S[i]))$$

But now we are back to the implication problem. We could change $A[i,j]$ above to $P[i,j]$, and turn the implication into an 'if and only if', but we then have the same problem as before linking $P[i,j]$ back to the adjacency matrix[1]. The problem is because we are introducing auxiliary variables, but only putting them on the right hand side of an implication, and thus setting the value of an auxiliary variable only partially constrains the decision variables. In terms of the implication operator, the auxiliary variables also need to appear on the left hand side of an implication, with a decision variable on the right (or on the right of a chain of implications).

6 Enforcing connectivity: the *connected-graph* global constraint

Instead of continuing to try different modelling primitives[2], we decided to implement a global constraint, which uses the CONNECTED-COMPONENTS(G) algorithm to update its internal data structures (the components) after each value assignment. The constraint takes the adjacency matrix, A, and the degree sequence, D, as input. It does no propagation, but will be violated if all the variables in A are instantiated and there is more than one component remaining. It requires three reversible data structures (reversible so that their values can be restored when the search process backtracks). C is a list of components, and each component is a list of integers representing vertices. P is an array maintaining for each vertex the index of its component in C. c is the number of components. To initialise, we create a unique component for each vertex. Whenever the search process assigns the value 1 to $A[i,j]$ (i.e selects the edge between i and j), where i and j were in different components, we update the data structures. We take the smaller of the two components, move all of its vertices into the larger, and update P for each of those vertices to point to the new component. Finally, we decrement c. When we reach a leaf node, if $c == 1$ then the graph is connected; if $c > 1$, then the graph is disconnected.

The data structures require $O(n)$ space, to store each vertex in a component, and to store the names of the components. If we assume that we always merge the shorter component into the larger, the updating requires at most $n/2$ operations to merge two components, and $n/2$ operations to update P, and thus is $O(n)$ at each node of the search tree. However, on a complete branch from root to leaf, we require $n-1$ updates, and thus $O(n^2)$ operations. This is the same cost as it would be to run the CONNECTED-COMPONENTS(G) algorithm afresh at each leaf node. In addition, however, we have the cost of updating the data structures on the branches that fail because of other constraints. Therefore, if this constraint is to be effective, we need to extend it by pruning or by detection of search nodes which have no connected realisations below them in order to save enough operations to account for the overhead.

[1]and we also found that Choco wouldn't let us do it anyway, reporting that the opposite of $\subseteq$ was not defined.

[2]although we have one more model, suggested by Ian Miguel, which we have not yet tried.

7 Adding propagation to *connected-graph*

We can improve the constraint by reasoning about the residual degrees of vertices and components during search, and by including some of the basic graph theory results. During a search, if vertex i has had k of its possible edges instantiated, then its *residual degree* is $degree(i) - k$. Let the residual degree of a component be the sum of the residual degrees of its vertices. The residual degree of a partially instantiated graph is the sum of the residual degrees over all vertices. To maintain information on the residual degrees, we need the following additional reversible data structures:

- an array RV of integers, maintaining the residual degree of each vertex. Each time we instantiate $A[i,j]$ to 1, we subtract 1 from $RV[i]$ and $RV[j]$.
- an integer r, maintaining the residual degree of the partial graph. Each time we instantiate any edge variable to 1, we subtract 2 from r (since each edge reduces two individual residual degrees by 1 each).
- an array RC of integers, maintaining the residual degree of each component. Each time we instantiate $A[i,j]$ to 1, we find the components p and q of i and j respectively using the array P. If they are the same component (i.e. $p == q$), then we subtract 2 from $RC[p]$; if they are different components, then we will merge them as before. Let p' be the merged component. We then set $RC[p'] = RC[p] + RC[q] - 2$.

We can identify a number of cases in which violations can be identified on initialisation:

1. if any vertex has initial degree of less than 1, and there is more than 1 vertex, then no connected graph is possible, since that vertex must be isolated;
2. if any vertex has an initial degree of more than $n-1$, then no graph is possible, since there are not enough other vertices with which to create the edges;
3. if the sum of the initial degrees is odd, then no graph is possible, by the handshaking lemma;
4. if the sum of the initial degrees is less than $2n-2$, then no connected graph is possible, since there are not enough edges to connect all the vertices;
5. if the sum of the initial degrees is greater than $n*(n-1)$, then no graph is possible, since there are not enough vertices to occupy all the edges;

We can also identify two cases for intermediate search nodes where the constraint must be violated, based on residual degree:

6. if the residual degree of a component drops to 0, and there is more than one component, then no completion of the partial graph can be connected, since all vertices in the component have used up all the edges, and none of those edges connect to the second component (by the definition of a component), then the first component can never become connected to the second;
7. if the residual degree of the graph drops to less than $2c-2$, where c is the number of components, then no

completion of the partial graph can be connected. This is by analogy to 4, in which we replace vertices by components - in order to ensure one component is connected to all the others, we will need to use at least one edge per remaining component (i.e. $c-1$ edges). Each edge contributes 2 to the residual degree, and therefore we need at least $2c-2$ edges to get a connected graph.

Finally, based on these violation checks, we can develop the following propagations:

8. if $n > 2$, then for all pairs of vertices i and j with initial degree of 1, force i and j to be not adjacent (since if we connect two vertices with degree of 1, then they must form an isolated component, and cannot be connected into a larger graph).

9. if $C[i]$ is a component with residual degree of 2, and there is more than one component, then if there is a pair j and k in $C[i]$ each with residual degree of 1, force j and k to be non-adjacent (if there is such a pair, then if we were to connect them together, there would be no more edges able to be instantiated incident on $C[i]$, and so $C[i]$ could not be connected to the rest of the graph). We apply this when $RC[i]$ is reduced to 2.

10. if $C[i]$ and $C[j]$ are two components with residual degree of 1, and there are more than two components, for the vertices v in $C[i]$ and w in $C[j]$ with residual degree of 1, force v and w to be non-adjacent (since each component must have exactly one vertex with residual degree > 0, and if we connect them, then the new combined component would have residual degree of 0, and so could not be connected to the rest of the graph). We apply this when $RC[i]$ is reduced to 1.

11. if the residual degree of the graph is $2c-2$, and there is more than one component, then for all components with residual degree greater than 1, force all pairs of vertices internal to the component to be non-adjacent (by the same analogy to 7, we need at least $c-1$ edges to connect up the components, and hence residual degree at least $2c-2$, so if we connect two vertices that are already in the same component, then we will not have enough edges remaining to connect up the other components). We apply this when r is first reduced to $2c-2$.

Propagation 8 is carried out at initialisation; the rest are carried out at nodes of the search tree.

The space requirement is still $O(n)$. The updates to the data structures are $O(n)$ as before, since the new updates each require only $O(1)$. The initialisation takes $O(n^2)$, because of 8. For propagation 9 we require at each search node at most n checks to find 2 vertices. For propagation 10, we require at most n checks to find both vertex v and all other vertices representing w. For propagation 11 there are at most $(n-1)^2$ pairs, and thus we require $O(n^2)$ checks. All four propagations only force values of 0, but are only triggered by variables being set to 1, and thus there is no cycle of propagators (although they may be invoked again if the other constraints set a variable to 1).

8 Experiments on *connected-graph*

We have implemented the adjacency matrix and the global *connected-graph* constraint in Ilog Solver 6.0. Each of the dead-end checks and propagations can be switched on or off independently. Recall that our purpose is not to generate all connected graphs as quickly as possible, but to develop a constraint that can be used with an external search procedure on problems with side constraints, to enforce connectivity. In particular, we have not considered symmetry, and there are many symmetries in these problems. We view symmetry as a separate feature, to be maintained independently from connectivity, and in other work we have begun to detect symmetries during the search [Wu, 2004]. However, we do want to evaluate the effectiveness of our model, and so we have tested it on pure connected graph generation problems. We have generated all possible degree sequences of lengths ranging from 6 to 10, with maximum vertex degree of 4. For each of these sequences, we then search for all possible solutions, and we have recorded for each length the total number of solutions (i.e. connected graphical realisations), the total number of backtracks-on-failure, and the total running time. We have run the algorithm with full propagation (*all*), with propagation 11 turned off (*-11*), with propagation only in the initialisation phase (*init*), with the odd degree initialisation filter and leaf node violations checks only (*even*), and with only checks at the leaf nodes (*leaf*) (i.e. no propagation and no other violation checks). We use the variable ordering heuristic `IloChooseMinSizeInt` (minimum domain), and a lexicographic value ordering. The experiments were carried out under Linux, with a 2.6 MHz processor. The results are presented in table 1.

Running with the leaf node violation checks only (*leaf*) is significantly slower than the four other methods which use some degree of filtering. However, we note that most of the improvement in running time for the other methods comes from *even*, the simple initialisation filter which fails sequences with an odd sum (which cannot have graphical realisations). The search with full propagation, *all*, is reducing the backtracks on failure by up to 10% compared to *even*, but is not significantly faster - in fact, for some of the smaller n, it is slightly slower. There could be a number of reasons for this. It is possible that our implementation is inefficient. Secondly, our propagations are relatively shallow - that is, they remove values which are likely to have been discovered at the next one or two depths in the tree, and so much of the work may be wasted. Finally, in this paper, we have only reasoned about residual degrees. We have not yet considered the consequences of setting an edge variable to 0 (i.e. rejecting the edge from the graph). We expect to be able to do more reasoning about the absence of edges to discover that subcomponents cannot be connected. However, even if we do improve our algorithms, when we compare the total fails for *even* with the total number of solutions, it appears that there is simply not that much propagation to be done - once we filter out those sequences of odd degree, only approximately 15% of the leaf nodes in the full search tree are not connected.

n	#	solutions		fails	time
6	84	703	all	193	0.14
			-11	193	0.14
			init	219	0.15
			even	259	0.13
			leaf	1243	0.18
7	120	10544	all	1811	0.37
			-11	1817	0.43
			init	2112	0.38
			even	2303	0.40
			leaf	18449	0.58
8	165	249569	all	38538	4.56
			-11	38604	4.48
			init	42512	4.46
			even	44010	4.51
			leaf	379152	8.25
9	220	7742661	all	1169783	127.19
			-11	1170429	127.08
			init	1230572	126.70
			even	1242061	127.03
			leaf	11764916	241.37
10	286	345052878	all	51550046	5717.91
			-11	51558645	5750.86
			init	52780580	5731.06
			even	52916767	5733.59
			leaf	478361894	10420.87

Table 1: finding all solutions for all degree sequences: n is the length of the sequence, # is the number of sequences of that length, *solutions* is the number of connected realisations, *fails* is the number of backtracks-on-failure for each method, and *time* is the total time in seconds for each method. Note that *solutions*, *fails* and *time* are the aggregated results over all sequences of the indicated length.

9 The Erdös-Gallai theorem

Since the solution density is so high in realisable sequences, it appears that graphicality may be more significant than connectivity, and so cheaper propagation to cut out non-graphical sequences, followed by leaf node checks on connectivity, might improve efficiency. We have therefore begun to investigate specific graphicality properties. The Erdös-Gallai theorem [Erdös and Gallai, 1960] states when a given degree sequence is *graphical*, i.e. under what conditions a graph can be produced with a given degree sequence. The theorem states that given a degree sequence $\sigma = d_1 \geq d_2 \geq ... \geq d_n$, this is graphical if and only if equation (1) holds for all $k < n$.

$$\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{n} min(k, d_i) \qquad (1)$$

This leads to the Havel-Hakimi algorithm [Havel, 1955; Hakimi, 1962] for a *realisation* of that sequence. We reproduce it below, in a version taken from [Gould, 1988], and it tests if the degree sequence σ is graphical.

1 If there exists an integer d in σ such that $d > n-1$ then halt and report failure. That is, we cannot have a vertex that is adjacent to more than $n-1$ other vertices.

2 If there are an odd number of odd numbers in σ halt and report failure. That is, there must be an even number of vertices of odd degree.

3 If the sequence σ contains a negative number then halt and report failure.

4 If the sequence σ is all zeros then halt and report success.

5 Reorder σ such that it is non-increasing.

6 Delete the first term d_1 from σ and subtract one from the next d_1 terms to form a new sequence. Go to step 3

Note that sequence $\sigma = 0, 0, 0, 0$ is graphical and realisable, and so there is nothing in the theorem or algorithm that states that the graph must be connected.

10 Using the Havel-Hakimi algorithm as a constraint

In the generation of graphs, edges are selected or rejected by the search process. Could the selection or rejection of an edge result in a dead end because equation (1) is violated? The answer is yes, and the (existence) proof follows.

Proof: Assume we have a degree sequence $S = 2, 1, 1$. This is graphical and can be realised as the path graph. Assume also that vertex v_1 has been constrained to have a degree of 2, and vertices v_2 and v_3 are to have a degree of 1. Further assume that the search process starts by selecting the edge (v_2, v_3). In the residual graph v_1 must have degree 2 and vertices v_2 and v_3 have a residual degree of 0. This is not a graphical sequence. $\square$

Therefore we should expect that our search process can generate dead-ends because equation (1) is violated, and early detection of this may result in reduced search effort. Note that step 2 of the above algorithm is redundant within the constraint encoding, i.e. if at the top of search the sequence σ contains an even number of odd numbers this will continue to be true during search. We prove this by considering 3 cases.

Proof: In case (1) search selects an edge (v, u) where the residual degree of v and u is even. When we add the edge we decrement the residual degrees and we now have two more vertices of odd degree. (2) search selects (v, u) and both vertices have an odd residual degree. When the edge is added we decrement the residual degrees and we remove two vertices of odd degree. (3) v has odd residual degree and u has even residual degree or conversely v has even residual degree and u has odd residual degree. We decrement both residual degrees and have the same number of vertices of even and of odd residual degree. Consequently such a constraint would serve no purpose. $\square$

It is worth noting that in [Mihail and Vishnoi, 2002] it is claimed that for a sequence to be graphical and potentially connected it is necessary and sufficient that (1) holds and that the sum of the degrees is at least $2(n-1)$, i.e. there are at least enough edges to produce a spanning tree. However, no algorithm is given for realising this other than to produce a spanning tree and then use the Havel-Hakimi algorithm on the residual graph.

n	nGSeq	nCSeq	Sol	nodes-	nodes+
4	11	6	9	6	9
5	31	19	61	62	62
6	102	68	787	1018	1017
7	342	236	15384	21329	21286
8	1213	863	580950	843812	841574

Table 2: Effect of the *graphical* constraint: *n* is the length of the sequence (with no maximum degree), *nGSeq* is the number of graphical sequences, *nCSeq* is the number of those that had connected realisations, *Sol* is the total number of connected realisations, *nodes-* is the number of nodes generated during the search without the *graphical* constraint, and *nodes+* is the number of nodes generated using the constraint.

We have coded up the Havel-Hakimi algorithm as a constraint *graphical*. The constraint takes as arguments an adjacency matrix of 0/1 constrained variables and a degree sequence. The constraint is tested whenever an edge is selected or rejected, i.e. a test is performed to determine if the residual graph continues to be graphical. If the residual graph is not graphical a contradiction is raised and a backtrack is forced. We have tested the effect of the constraint, and the resuls are shown in Table 2. Note that the results in this table are not comparable to the results in Table 1, since we have used a different experimental set-up. In particular, the sequences now have no maximum degree, only graphical initial sequences were considered, and a different connectivity filter was applied during search. From the results, we can see that the *graphical* constraint does propagate, even in the search for realisations of sequences that are initially graphical, and that the effect does increase as we move to larger problems.

11 Conclusion and Future work

We have described some explorations of constraint programming in graph generation, concentrating on forcing the graphs to be connected. We have presented a straightforward encoding of the simple graph generation problem. We have briefly described two failed attempts to model the connectivity constraint using standard modelling primitives. We have discussed a new global constraint, *connected-graph*, and developed some violation checks and propagations. We have tested the constraint on some pure generation problems, and we have shown that the constraint does reduce search time. However, perhaps because of the very high solution density, almost all of the efficiency gains come from a filter rejecting non-graphical degree sequences, rather than from reasoning about connectivity. We have then discussed an alternative constraint, *graphical*, which simply enforces graphicality.

As stated in the introduction, the problem of enforcing connectivity during graph generation has practical applications, and we are continuing to develop constraint-based solutions for those applications. Although there does not appear to be much scope for propagation in the pure problem, based on the ratio of the number of solutions to the total leaf nodes, we intend to develop better propagators based on reasoning about the rejection of an edge, and on other methods based on graph theory. Our first step will be to unify the different methods discussed in this paper. We will also compare the degree-sequence model to the cardinality matrix constraint [Régin and Gomes, 2004], which could be used to enforce a degree sequence. The most scope for improvement in the application problems is in symmetry detection, and we have begun to categorize and break the symmetries in the graphs.

12 Acknowledgments

We thank Ian Miguel and Barbara Smith for comments on an earlier version of this paper. We are grateful for support from Science Foundation Ireland ((00/PI.1/C075), Enterprise Ireland (SC/2003/81) and Ilog SA.

References

[Cormen *et al.*, 2001] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. 2001.

[Erdös and Gallai, 1960] P. Erdös and T. Gallai. Graphs with prescribed degrees of vertices. *Mat. Lapok*, 11:264–274, 1960.

[Gould, 1988] R. J. Gould. *Graph Theory*. 1988.

[Hakimi, 1962] S.L. Hakimi. On the realization of a set of integers as degrees of the vertices of a graph. *J. SIAM Appl. Math.*, 10:496–506, 1962.

[Havel, 1955] V. Havel. A remark on the existence of finite graphs. *Casopis Pest. Mat.*, 80:477–480, 1955.

[Le Pape *et al.*, 2002] C. Le Pape, L. Perron, J.-C. Régin, and P. Shaw. Robust and parallel solving of a network design problem. In *CP2002 (ed. P. van Hentenryck), LNCS 2470*, pages 633–648, 2002.

[Mihail and Vishnoi, 2002] M. Mihail and N. K. Vishnoi. On generating graphs with prescribed vertex degrees for complex network modelling. In *ARACNE 2002*, pages 1–11, 2002.

[Pesant and Soriano, 2002] G. Pesant and P. Soriano. An optimal strategy for the constrained cycle cover problem. *Annals of Mathematics and Artificial Intelligence*, 34:313–325, 2002.

[Régin and Gomes, 2004] J.-C. Régin and C. Gomes. The cardinality matrix constraint. In *CP2004 (ed. M. Wallace), LNCS 3258*, pages 572–587, 2004.

[Shiloach, 1981] Y. Shiloach. Another look at the degree constrained subgraph problem. *Inf. Proc. Letters*, 12:89–92, 1981.

[Simonis, 2004] H. Simonis. Constraint applications using graph theory results. CPAIOR 2004 masterclass, 2004. http://www.icparc.ic.ac.uk/ hs/.

[Sorlin and Solnon, 2004] S. Sorlin and C. Solnon. A global constraint for graph isomorphism problems. In *CPAIOR 2004, LNCS 3011*, pages 287–301, 2004.

[Wu, 2004] C. W. Wu. Modelling chemical reactions using constraint programming and molecular graphs. In *CP2004 (ed. M. Wallace), LNCS 3258*, page 808, 2004.

An n-ary Constraint for the Stable Marriage Problem[*]

Chris Unsworth, Patrick Prosser
Department of Computing Science
University of Glasgow, Scotland
{chrisu,pat}@dcs.gla.ac.uk

Abstract

We present an n-ary constraint for the stable marriage problem. This constraint acts between two sets of integer variables where the domains of those variables represent preferences. Our constraint enforces stability and disallows bigamy. For a stable marriage instance with n men and n women we require only one of these constraints, and the complexity of enforcing arc-consistency is $O(n^2)$ which is optimal in the size of input. Our computational studies show that our n-ary constraint is significantly faster and more space efficient than the encodings presented in [3]. We also introduce a new problem to the constraint community, the sex-equal stable marriage problem.

1 Introduction

In the Stable Marriage problem (SM) [2; 5] we have n men and n women. Each man ranks the n women into a preference list, as do the women. The problem is then to produce a matching of men to women such that it is stable. By a matching we mean that there is a bijection from men to women, and by stable we mean that there is no incentive for partners to divorce and elope. A matching is unstable if there are two couples (m_i, w_j) and (m_k, w_l) such that m_i prefers w_l to his current partner w_j, and w_l prefers m_i to her current partner m_k.

Figure 1 is an instance of the stable marriage problem, and has 6 men and 6 women. Figure 1 shows the problem initially, with each man and woman's preference list. Figure 2 shows the intersection of the male and female-oriented Gale-Shapley lists (GS-lists) [5], where the GS-lists are reduced preference lists. A man-optimal (woman-pessimal) stable matching can now be found by marrying men (women) to their most (least) preferred choices in there GS-lists. Conversely, we can produce a woman-optimal (man-pessimal) matching by marrying women (men) to their most (least) preferred choice in their GS-lists. An instance of SM admits at least one stable matching and this can be found via the Extended Gale-Shapley algorithm in time $O(n^2)$, where there are n men and n women.

[*]The first author is supported by EPSRC. Software support was given by an ILOG SA's academic grant.

Men's lists	Women's lists
1: 1 3 6 2 4 5	1: 1 5 6 3 2 4
2: 4 6 1 2 5 3	2: 2 4 6 1 3 5
3: 1 4 5 3 6 2	3: 4 3 6 2 5 1
4: 6 5 3 4 2 1	4: 1 3 5 4 2 6
5: 2 3 1 4 5 6	5: 3 2 6 1 4 5
6: 3 1 2 6 5 4	6: 5 1 3 6 4 2

Figure 1: An SM instance with 6 men and 6 women

Men's lists	Women's lists
1: 1	1: 1
2: 2	2: 2
3: 4	3: 4 6
4: 6 5 3	4: 3
5: 5 6	5: 6 4 5
6: 3 6 5	6: 5 6 4

Figure 2: the corresponding GS-lists

We present a simple constraint encoding for the stable marriage problem. We introduce a specialised n-ary constraint with only three methods, where each method is no more than six lines of code. We show how enforcing arc-consistency in this encoding results in the male-oriented Gale-Shapley lists. This minimal encoding cannot be used in search and only achieves directed arc-consistency, from men to women. We then go on to show how we can extend this encoding by introducing a modest amount of additional code, such that the encoding can be used in search, can be embedded in richer impure problems where the stability of marriages is only part of a larger problem, and the male and female oriented GS-lists are produced. Our empirical results suggest, that although our encodings has $O(n^2)$ time complexity, the same as the optimal encoding proposed in [3], our constraint significantly outperforms this encoding in both space and time.

2 The Extended Gale-Shapley Algorithm (EGS)

We now describe the male-oriented Extended Gale-Shapley (EGS) algorithm (shown in Figure 3). In particular, we explain what is meant by a *proposal*, an *engagement*, and for a man to become *free*. We will use this later to show that this algorithm and our constraint encoding are equivalent.

The EGS algorithm [5] produces a stable matching between men m_1 to m_n and women w_1 to w_n, where each man (woman) ranks each of the women (men) into preference order. Via a process of proposals from men to women the algorithm delivers reduced preference lists, called GS-lists (Gale-Shapley lists), such that if each man (woman) is paired with his (her) best (worst) partner in their GS-list the marriages will be stable.[1]

```
 1  assign each person to be free
 2  WHILE (some man m is free)
 3  DO BEGIN
 4     w := first woman on m's list
 5     IF (some man p is engaged to w)
 6     THEN assign p to be free
 7     assign m and w to be engaged
 8     FOR (each successor p of m on w's list)
 9     DO BEGIN
10        delete p from w's list
11        delete w from p's list
12        END
13     END
```

Figure 3: The male-oriented Extended Gale/Shapley algorithm.

We will assume that we have an instance I of the stable marriage problem, and that for any person q in I, $PL(q)$ is the ordered list of persons in the original preference list of q and $GS(q)$ is the ordered list of people in the GS-list for q, and initially $GS(q)$ equals $PL(q)$. In a *proposal* from man m to woman w, w will be at the head of the man's GS-list $GS(m)$. This leads to an *engagement* where m is no longer free and all men that w prefers less than m are removed from her GS-list, i.e. the last entry in $GS(w)$ becomes m. Further, when a man p is removed from $GS(w)$ that woman is also removed from his GS-list, i.e. w is removed from $GS(p)$, consequently bigamy is disallowed. Therefore m and w are engaged when m is no longer free, w is head of $GS(m)$, and m is at the tail of $GS(w)$. A man p becomes *free* when p was engaged to w (i.e. the head of $GS(p)$ is w) and w receives a proposal from man m that she prefers to p. On becoming free, p is added to the list of free men and w is removed from $GS(p)$.

The algorithm starts with all men free and placed on a list (line 1). The algorithm then performs a sequence of proposals (lines 2 to 13). A man m is selected from the free list (line 2), and his most preferred woman w is selected (line 4). If w is engaged, then her partner p becomes free. The pair m and w then become engaged (lines 7 to 12).

3 Preliminaries

We assume that the men and women's preference lists have been read into two 2-dimensional integer arrays mpl and wpl respectively. $mpl[i]$ is the preference list for the i^{th} man where $mpl[i][j]$ is the ith man's jth preference, and similarly $wpl[j]$ is the preference list for the j^{th} woman. Using our problem in Figure 1, if we consider our 3^d man he will have a preference list $mpl[3] = (1, 4, 5, 3, 6, 2)$.

We also assume we have the inverse of the preference lists, i.e. mPw and wPm, where $mPw[i][j]$ is the i^{th} man's preference for the j^{th} woman and $wPm[k][l]$ is the kth woman's preference for the lth man. Again, considering the 3^d man in Figure 1, his inverse preference list will be $mPw[3] = (1, 6, 4, 2, 3, 5)$, $mPw[3][2]$ is his preference for the 2nd woman, and that is 6, i.e. woman 2 is in the 6th position of man 3's preference list.[2]

We associate a constrained integer variable with each man and each woman, such that $x[i]$ is a constrained integer variable representing the i^{th} man m_i in stable marriage instance I and has a domain $dom(x[i])$ initially of 1 to n. Similarly, we have an array of constrained integer variables for women, such that $y[j]$ represents the j^{th} woman w_j in I. The values in the domain of a variable correspond to preferences, such that if variable $x[i]$ is assigned the value a this corresponds to m_i being married to his a^{th} choice of woman, and this will be woman $mpl[i][a]$. For example, if $x[2]$ (in Figure 1) is set to 3 then this corresponds to m_2 marrying his 3^d choice, w_1 (and conversely $y[1]$ would then have to be assigned the value 5). Again referring to Figure 1 our 6th man's domain is $dom(x[6]) = (1, 2, 3, 4, 5, 6)$, as is everyone else's, and in Figure 2 $dom(x[6]) = (1, 4, 5)$. We also assume that we have the following functions, each being of $O(1)$ complexity, that operate over constrained integer variables:

- $getMin(v)$ delivers the smallest value in $dom(v)$.
- $getMax(v)$ delivers the largest value in $dom(v)$.
- $getVal(v)$ delivers the instantiated value of v.
- $setMax(v, a)$ sets the maximum value in $dom(v)$ to be min$(getMax(v), a)$.
- $setVal(v, a)$ instantiates the variable v to the value a.
- $remVal(v, a)$ removes the value a from $dom(v)$.

We assume that constraints are processed by an arc-consistency algorithm such as AC5 [9] or AC3 [7]. That is, the algorithm has a stack of constraints that are awaiting revision and if a variable loses values then all the constraints that the variable is involved in are added to the stack along with the method that must be applied to those constraints, i.e. the stack contains methods and their arguments. Furthermore, we also assume that a call to a method, with its arguments, is only added to the stack if it is not already on the stack. We'll refer to this stack as the *call stack*.

4 An n-ary Stable Marriage Constraint (SM2N)

We now give a description of our n-ary stable marriage constraint, where arc-consistency on such an encoding is equivalent to an application of the male-oriented EGS algorithm.

[1]Strictly speaking, the given algorithm produces MGS-lists, the male GS-lists. But for the sake of brevity we will refer to them as GS-lists.

[2]The inverse of the preference lists can be created when reading in the preference lists such that $mPw[i][mpl[i][j]] = j$, and this does not affect the overall complexity of constructing our model.

Note that the constraint as described minimally cannot be used within a search process, however we will later show how this can be done. Our constraint is n-ary in that it constrains n men and n women such that stability is maintained and bigamy is disallowed, although it achieves only 2-consistency.[3] In a stable marriage problem with n men and n women we will then require only one of these constraints. We now start by describing the attributes of the constraint and the three methods that act upon it. We will use a java-like pseudo-code such that the . (dot) operator is an attribute selector, such that $a.b$ delivers the b attribute of a.

4.1 The attributes

A n-ary stable marriage constraint (SM2N) is an object that acts between n men and n women, and has the following attributes:

- x and y are constrained integer variable arrays representing the men and women that are constrained, such that $x[i]$ is the constrained integer variable corresponding to m_i and $y[j]$ corresponds to w_j.
- xpl and ypl are 2-dimensional integer arrays which contain the male and female preference lists respectively, such that $xpl[i]$ equals $PL(m_i)$ and $xpl[i][j]$ contains m_i's j^{th} choice woman.
- xPy and yPx are 2-dimensional integer arrays which contain the male and female inverse preference lists respectively, such that $xPy[i][j]$ contains man i's preference for w_j.
- yub is an array of integer variables which contain the previous upper bounds of all y variables. All are set to n at the start of search and are updated by the deltaMax(i) method detailed below.

4.2 The propagation methods

We now describe three methods that achieve male-oriented arc-consistency.

deltaMin(i)
This method is called when the lower bound of $dom(x[i])$ increases. The lower bound of $dom(x[i])$ increasing signifies that m_i has been rejected by his favourite choice of partner and thus must propose to his new favourite available partner. To do this we first find m_i's favourite available partner w_j (line 2), then remove all men from the list of w_j she likes less than m_i (line 3).

```
1.   deltaMin(i)
2.    j = xPy[i][getMin(x[i])]
3.    setMax(y[j],yPx[j][i])
```

deltaMax(j)
This method is called when the upper bound of $dom(y[j])$ is reduced. To maintain consistency w_j needs to be removed from the domains of all men that have been removed from her domain. This is done by looping once for each value that has been removed from the tail of $dom(y[j])$ since the last call to deltaMax(j) (line 2). Within the loop a m_i that has been removed from $dom(y[j])$ is selected (line 3) and then w_j is removed from $dom(x[i])$. When all relevant men have had their domains' altered (line 5) yub is updated (line 6).

```
1.   deltaMax(j)
2.    FOR (k = getMax(y[j])+1 to yub[j])
3.     i = yPx[j][k]
4.     remVal(x[i],xPy[i][j])
5.    END FOR LOOP
6.    yub[j] = getMax(y[j])
```

init()
The $init$ method is called when the constraint is created, and is simply a call to $deltaMin$ for each of the n men variables.

```
1.   init()
2.    FOR (i = 1 to n)
3.     deltaMin(i)
4.    END FOR LOOP
```

5 Comparison to EGS

We now compare the behaviour of our n-ary constraint model (SM2N) to the male-oriented EGS algorithm. In our comparison we will describe steps in the EGS algorithm in *italics* and the SM2N constraint encoding in normal font. Sometimes we will use m and w as a particular person (rather than m_i and w_j), and x and y as particular variables (rather than $x[i]$ and $y[j]$) for sake of brevity. Additionally, we assume we have the function $fiance(y[i])$ and that it delivers the integer k where $k = wpl[i][max(dom(y[i]))]$, i.e. $x[k]$ is the least preferred partner of $y[i]$.

- *Initially the EGS algorithm sets all men to be free by adding them to the free list (line 1).* Equivalently, when propagation starts the call to $init()$ will cause the set of calls $\{deltaMin(i)|1 \leq i \leq n\}$ to be added to the empty call stack.
- *EGS picks a man m from the free list and he then proposes to his first choice woman w (lines 4 to 7).* Initially the call stack will contain n calls to the $deltaMin$ method, called directly via $init$. When executing the call $deltaMin(i)$, man $x[i]$ will make the equivalent of a proposal to his first choice woman (as described next).
- *When m makes a proposal to w all values that appear in $GS(w)$ after the proposing man are removed (lines 8 to 10), i.e. they become engaged.* When the call $deltaMin(i)$ is made, where $y[j]$ is $x[i]'s$ favourite, the maximum of $dom(y[j])$ is set to $y[j]'s$ preference for $x[i]$, therefore removing all less preferred men. Effectively, $x[i]$ and $y[j]$ become engaged.
- *To maintain monogamy EGS removes the newly engaged woman from the GS-lists of all men that have just been removed from her preference list (line 11).* From the action above, the maximum of $dom(y[j])$ has been lowered, consequently a call to $deltaMax(j)$ will be added to the call stack. In that call to $deltaMax(j)$, $y[j]$ is removed from $dom(x[k])$ for all k where k has been removed from the tail of $dom(y[j])$. Therefore, $x[k]$ and $y[j]$ can never be married.

[3]A detailed explanation of just what we mean by 2-consistency in this model is given in section 6.

- *In EGS, if m makes a proposal to w, who is already engaged to p, then $w's$ previous fiance p is assigned to be free and added to the free list (lines 5 and 6.)* On initiating the call $deltaMin(i)$ where $y[j]$ is $x[i]'s$ favourite available woman, $y[j]'s$ fiance corresponds to the maximum value in $dom(y[j])$, because all less preferred men will have been removed (as above). Therefore if $y[j]$ receives a proposal from $x[i]$ via the call $deltaMin(i)$, and $y[j]$ prefers $x[i]$ to her current fiance $x[k]$ (where $k = fiance(y)$) the maximum of $dom(y[j])$ will be set lower than her preference for $x[k]$ and therefore her preference for $x[k]$ will be removed from $dom(y[j])$. Consequently, the call $deltaMax(j)$ will then be put on the call stack, which will remove $x[k]'s$ preference for $y[j]$ from $dom(x[k])$. Because $y[j]$ was $x[k]'s$ previous favourite, $x[k]'s$ preference for $y[j]$ would have been $min(dom(x[k]))$. Therefore removing that value will increase $x[k]'s$ domain minimum, and the call $deltaMin(k)$ will then be added to the stack. And this effectively assigns man $x[k]$ to be free.

6 Arc-consistency in the Model

On the completion of arc-consistency processing, the variable domains can be considered as $GS - domains$. That is, $a \in dom(x[i]) \leftrightarrow w_j \in GS(m_i) \wedge j = mpl[i][a]$. Furthermore, $b \in dom(y[j]) \leftrightarrow m_i \in GS(w_j) \wedge i = wpl[j][b]$.

The GS-domains are 2-consistent such that if man m_i is married to a woman w_j (i.e. $x[i] = a \wedge a \in dom(x[i]) \wedge j = mpl[i][a]$) then any woman w_l can then marry some man m_k without forming a blocking pair or a bigamous relationship. That is, for an arbitrary woman w_l there exists a value $b \in dom(y[l])$ such that $k = wpl[l][b] \wedge (mPw[i][j] < mPw[i][l] \vee wPm[l][k] < wPm[l][i]) \wedge i \neq k \wedge j \neq l$. Furthermore if a man m_i is married to a woman w_j then any other man m_k can then marry some woman w_l, where $l \neq j$.

It is important to note, that although our constraint is n-ary it only achieves 2-consistency. It is our opinion that the cost of achieving a higher level of consistency would be of little advantage. This is so because by maintaining 2-consistency, and using a suitable value ordering heuristic in the model during search we are guaranteed failure-free enumeration of all solutions [3].

In [5] Theorem 1.2.2 it is proved that all possible executions of the Gale-Shapley algorithm (with men as proposers) yield the same stable matchings. Our encoding mimics the EGS algorithm (as shown in section 5) and we claim (without proof) that the encoding reaches the same fixed point for all ordering of the revision methods on the call stack.

7 Complexity of the model

In [5] section 1.2.3 it is shown in the worst case there is at most $n(n-1)+1$ proposals that can be made by the EGS algorithm, and that the complexity is then $O(n^2)$. We argue that the complexity of our SM2N encoding is also $O(n^2)$. First we claim that the call to our method $deltaMin()$ is of complexity $O(1)$. The $deltaMax()$ method is of complexity $O(r)$, where r is the number of values removed from the tail of variable since the last call to $deltaMax()$ for this variable.

Because there are n values in the domain of variable y the worse case complexity for all possible calls to $deltaMax(j)$ is $O(n)$. Equally there are n values in the domain of variable x and thus the worse case complexity for all possible calls to $deltaMin(i)$ is $O(n)$. Therefore because there are n y variables and n x variables, the total worst case complexity for all possible calls to $deltaMin(i)$ and $deltaMax(j)$ is $O(n^2)$.

8 Enhancing the model

The full GS-Lists are the union of the male and female Gale-Shapley lists remaining after executing male and female oriented versions of EGS. It has been proven that the same lists can be produced by running the female orientated version of EGS on the male-oriented GS-lists [5]. Because SM2N produces the same results as EGS the full GS-Lists can be produced in the same way. But because of the structure of this specialised constraint it is also possible to combine the male and female orientated versions of SM2N into one constraint. This combined gender free version of SM2N will then produce the full GS-List with only one run of the arc-consistency algorithm. To create the gender free version all of the methods presented in this paper must then be symmetrically implemented from the male and female orientations.

The SM2N constraint as presented so far has only considered domain values being removed by the constraint's own methods. If we were to use the constraint to find all possible stable matchings, unless arc consistency reduces all variable domains to a singleton, it will be necessary to assign and remove values from variable domains as part of a search process. Therefore, we need to add code to SM2N to maintain consistency and stability in the event that domain values are removed by methods other than those within SM2N. It is important to note that these external domain reductions could also be caused by side constraints as well as a search process.

There are four types of domain reduction that external events could cause: a variable is instantiated; a variable's minimum domain value is increased; a variable's maximum domain value is reduced; one or more values are removed from the interior of a variable's domain. We now describe two additional methods, $inst$ and $removeValue$, and the enhancements required for $deltaMin$. We note that $deltaMax$ does not need to change, and describe the required enhancements for incomplete preference lists.

inst(i)

The method $inst(i)$ is called when a variable $x[i]$ is instantiated.

```
 1.   inst(i)
 2.    For (k = 0 to getVal(x[i])-1)
 3.     j = xPy[i][k]
 4.     setMax(y[j],yPx[j][i]-1)
 5.    END FOR LOOP
 6.    j = xPy[i][getVal(x[i])]
 7.    setVal(y[j],yPx[j][i])
 8.    For (k = getVal(x[i])+1 to n)
 9.     j = xPy[i][k]
10.     remVal(y[j],yPx[j][i])
11.    END FOR LOOP
```

This method removes all values from the set of y variables to prevent variable $x[i]$ being involved in a blocking pair or inconsistency. To prevent $x[i]$ from creating a blocking pair, all the values that corresponds to men less preferred than $x[i]$, are removed from the domains of all women that $x[i]$ prefers to his assigned partner (lines 2-5). Since $x[i]$ is matched to $y[j]$, $y[j]$ must now be matched to $x[i]$ (lines 6,7). To maintain consistency $x[i]$ is removed from the domains of all other women (lines 8-11)). The complexity of this method is $O(n)$ and because there are n x variables and each can only be instantiated once during propagation, the total time complexity of all possible calls to $inst(i)$ is $O(n^2)$.

removeValue(i,a)

This method is called when the integer value a is removed from $dom(x[i])$, and this value is neither the largest nor smallest in $dom(x[i])$.

```
1.   removeValue(i,a)
2.    j = xPy[i][a]
3.    remVal(y[j],yPx[j][i])
```

The woman the value a corresponds to is found (line 2) then $x[i]$ is removed from her domain (line 3), and this must be done to prevent bigamy.

Enhancements to deltaMin(i)

Up till now we have assumed that all values removed from the head of $dom(x[i])$ are as a result of m_i being rejected by some w_j. We now drop this assumption in the following enhanced version. In this method we add a new variable array named xlb, and this is similar to the yub array except it holds the previous lower bound of x. All elements in xlb are initialised to 1 and are updated and used only by the $deltaMin$ method.

```
1.   deltaMin(i)
2.    j = xPy[i][getMin(x[i])]
3.    setMax(y[j],yPx[j][i])
4.    FOR (k = xlb[i] to getMin(x[i])-1)
5.     j = xPy[i][k]
6.     setMax(y[j],yPx[j][i]-1)
7.    END FOR LOOP
8.    xlb[i] = getMin(x[i])
```

Lines 1 to 3 are as the original. The next four lines (lines 4-7) cycle through each of the values that have been removed from the head of $dom(x[i])$ since the last call to $deltaMin(i)$ (line 4). $y[j]$, which the removed value corresponds to, is then found (line 5), and then all values that are not strictly greater than her preference for $x[i]$ are removed from $dom(y[j])$ (line 6). The lower bound of the man variable $x[i]$ is then updated (line 8).

No enhancements to deltaMax(j)

We now consider the situation where some process, other than a proposal, removes values from the tail of $dom(y[j])$, i.e. when the maximum value of $dom(y[j])$ changes. The $deltaMax$ method will be called, and the instance continues to be stable as all values remaining in $dom(y[j])$ corresponding to men w_j prefers to the removed values. However, we need to prevent bigamy, by removing w_j from the corresponding $dom(x)$ variables removed from the tail of $dom(y[j])$, and this is just what $deltaMax$ does. Therefore, no enhancement is required.

Incomplete Lists (SMI)

The encoding can also deal with incomplete preference lists, i.e. instances of the stable marriage problems with incomplete lists (SMI). For a SM instance of size n we introduce the value $n+1$. The value $n+1$ must appear in the preference lists $mpl[i]$ and $wpl[j]$ as a *punctuation* mark, such that any people after $n+1$ are considered unacceptable. For example, if we had an instance of size 3 and a preference list $PL(m_i)$ = (3,2) we would construct $mpl[i] = (3, 2, 4, 1)$ and this would result in the inverse $mPw[i] = (4, 2, 1, 3)$. Consequently $x[i]$ would always prefer to be unmatched (assigned the value 4) than to be married to $y[1]$. We now need to modify the $init$ method such that it sets the maximum value in $dom(x[i])$ to be $mPw[i][n+1]$. These modifications will only work in the full implementation (i.e. it requires the above enhancements).

Reversible integers

In this encoding we have used two variable arrays which contain dynamic data. yub and xlb are initialised to n and 1 respectively, but these values will be updated as the problem is being made arc-consistent. If we are only looking for the first solution then we need only use normal integers to hold these values. However, when the constraint solver backtracks and values that had been removed from the domain of a variable are reintroduced then the values held in yub and xlb will no longer be correct. To fix this problem we have to tell the solver that when it backtracks it needs to reverse the changes to yub and xlb as well as the variables domains. This is done by using a reversible integer variable. This class should be supplied in the constraint solver toolkit. The solver will then store the values of each of the reversible variables at each choice point and restore them on backtracking.

9 Computational Experience

We implemented our encodings using the JSolver toolkit [1], i.e. the Java version of ILOG Solver. In a previous paper [8] we presented a specialised binary constraint (SM2) for the stable marriage problem, and presented some results comparing the SM2 constraint with the two constraint encoding in [3]. Here we show a chopped down version of those results, with the results obtained by running SM2N on the same set of test data included. The other model shown in the results table is the optimal boolean encoding (Bool) as presented in [3]. Our experiments were run on a Pentium 4 2.8Ghz processor with 512 Mbytes of random access memory, running Microsoft Windows XP Professional and Java2 SDK 1.4.2.6 with an increased heap size of 512 Mbytes.

	size n					
model	100	200	400	600	800	1000
Bool	1.2	4.4	ME	ME	ME	ME
SM2	0.23	0.5	1.82	4.21	8.02	12.47
SM2N	0.02	0.06	0.21	0.51	0.95	2.11

Table 1: Average computation times in seconds to produce the GS-lists, from 10 randomly generated stable marriage problems each of size n

Our first experiment measures the time taken to generate a model of a given SM instance and make that model arc-consistent, i.e. to produce the GS-lists. Table 1 shows the average time taken to produce the GS-lists for ten randomly generated instances of size 100 up to 1000. Time is measured in seconds, and an entry ME means that an out of memory error occurred. We can see that the SM2N constraint dominates the other models.

	size n					
model	100	200	400	600	800	1000
Bool	2.02	6.73	ME	ME	ME	ME
SM2	0.47	1.97	10.13	27.27	54.98	124.68
SM2N	0.03	0.07	0.24	0.73	1.56	3.35

Table 2: Average computation times in seconds to find all solutions to 10 randomly generated stable marriage problems each of size n

This second experiment measures the time taken to generate a model and find all possible stable matchings. Table 2 shows the average time taken to find all solutions on the same randomly generated instances used in the first experiment. Again it can be seen that the SM2N model dominates the other models. In summary, when the boolean encoding solves a problem the n-ary constraint does so nearly 100 times faster, and the n-ary constraint can model significantly larger problems than the boolean encoding.

Tables 1 and 2 raise the following question, if the Bool encoding is optimal then why is it dominated by the SM2 encoding, when SM2 is $O(n^3)$ time and the Bool encoding is $O(n^2)$ time? The main reason for this is that there is no significant difference in the space required to represent variables with significant differences in domain size, because domains are represented as intervals when values are consecutive. Considering only the variables, the Bool encoding uses $O(n^2)$ space whereas the SM2 model uses $O(n)$ space. For example, with $n = 1300$ the Bool encoding runs out of memory just by creating the 2.1300^2 variables whereas the SM2 model takes less than 0.25 seconds to generate the required 2600 variables each with a domain of 1 to 1300. Theoretically the space complexity of the constraints used by SM2 and Bool are the same. In practise this is not the case as SM2 requires exactly n^2 constraints to solve a problem of size n whereas Bool requires $2n + 6n^2$ constraints. Therefore the Bool encoding requires more variables and more constraints, resulting in a prohibitively large model. The same argument also applies to the performance of the SM2N constraint, i.e. the n-ary constraint is more space efficient that the Bool encoding, is of the same time complexity, and this results in superior performance. The space and time complexities of these models are tabulated below. Note that the $O(n^2)$ constraint-space for SM2N is a consequence of the storage of the preference lists and their inverses.

This Third experiment shows how SM2N can handle larger problems. Table 4 shows the average time taken to both produce the GS-Lists and find all solutions for one hundred randomly generated instances of size 1000 up to 2000, again the times are in seconds.

	Bool	SM2	SM2N
time	$O(n^2)$	$O(n^3)$	$O(n^2)$
constraints space	$O(n^2)$	$O(n^2)$	$O(n^2)$
variables space	$O(n^2)$	$O(n)$	$O(n)$

Table 3: Summary of the complexities of the three SM constraint models

	size n					
problem	1000	1200	1400	1600	1800	2000
AC	2.11	3.12	5.93	8.71	11.59	20.19
All	3.35	5.09	8.8	12.92	18.96	26.81

Table 4: Average computation times in seconds from 100 randomly generated stable marriage problems each of size n

10 Sex equal optimisation

The sex equal stable marriage problem (SESMP) as posed in [5] as an open problem, is essentially an optimisation problem. A male optimal solution to an SMP is where all men get there best possible choices from all possible stable matchings (and all women get there worst), and in a woman optimal solution all women are matched to there best possible choices (and all men to there worst). A sex equal matching is where both the men and the women are equally well matched. This problem has been proven to be NP-Hard [6].

In a $SESMP$ all men will have a score for each woman and all women will have a score for each man, man m_i's score for woman w_j is $mScore[i][j]$ and woman w_j's score for man m_i is $wScore[j][i]$. In an unweighted $SESMP$ all scores will be the same as the preferences, so $mScore[i][j]$ would equal $mPw[i][j]$ and $wScore[j][i]$ would equal $wPm[j][i]$. In a weighted $SESMP$ this is not so, but the same ordering must be maintained meaning $mScore[i][j] < mScore[i][k]$ iff $mPw[i][j] < mPw[i][k]$. For any matching M all men and women will score the matching determined by which partner they are match to in M. If man m_i is matched to woman w_j in matching M then m_i will give that matching a score of $mScore[i][j]$ and woman w_j will give it a score of $wScore[j][i]$. The sum of all scores given by men for a matching M equals $sumM(M)$ and the sum of the women's scores is $sumW(M)$. A matching M for an instance I of the stable marriage problem is sex equal iff there exists no matching M such that the absolute difference between the $sumM(M)$ and $sumW(M)$ is less than the absolute difference between $sumM(M)$ and $sumW(M)$.

Because the values in the domains of the x and y variables are preferences, it makes finding an unweighted sex equal matching with $SM2N$ simple. All that is required is to add a search goal to minimise the absolute difference between the sum of all x variables and the sum of all y variables. We tested this using the same test data as in Table 4 and the results are tabulated below. These results can be compared to those in Figure 6 of [8], where the Bool encoding failed to model problems with 300 or more men and women, and at $n = 1000$ the SM2 model was more than 15 times slower than the SM2N model. We believe that this demonstrates the versatility of our constraint, in that we can easily use the constraint as part of a richer problem.

	size n					
problem	1000	1200	1400	1600	1800	2000
SE	3.65	5.02	8.73	14.44	17.59	22.44

Table 5: Average computation times in seconds to find all solutions to 100 randomly generated sex-equal stable marriage problems, each of size n, modelled using the SM2N constraint.

11 Implementation

The SM2N constraint was originally developed using the choco constraints tool kit, and the way the constraint has been introduced reflects that. In choco to implement a user defined constraint, the $abstractLargeIntConstraint$ class is extended. This class contains the methods $awake$, $awakeOnInf$, $awakeOnSup$, $awakeOnRem$ and $awakeOnInst$. These methods are the equivalent of the ones used to introduce the constraint. $awake$ is the same as $init$, $awakeOnInf$ and $awakeOnSup$ are the same as $deltaMin$ and $deltaMax$ and $awakeOnInst$ is the same as $inst$. To implement a constraint in Ilog JSolver we first state when the constraint needs to be propagated, i.e. when a domain value is removed, when the range changes (meaning the upper or lower bound changes) or just when a variable is instantiated. We then need to define a method that will handle propagation when such an event occurs. For the SM2N constraint we stated it was to be propagated every time the range of a variable changed. We then used conditional statements to ascertain which bound had changed, and used the methods as presented above to handle the propagation.

12 Conclusion

We have presented a specialised n-ary constraint for the stable marriage problem, possibly with incomplete lists. The constraint can be used when stable marriage is just a part of a larger, richer problem. Our experience has shown that this constraint can be implemented in a variety of constraint programming toolkits, such as JSolver, JChoco, and Koalog. The complexity of the constraint is $O(n^2)$. Although this is theoretically equal to the optimal $O(n^2)$ complexity of the Boolean encoding in [3], our constraint is more practical, typically being able to solve larger problems faster. For example, we have been able to enumerate all solutions to instances of size 2000 in seconds, whereas in [4] the largest problems investigated were of size 60. We have also presented the first study of SESMP using a constraint solution, i.e. where the stable matching constraints are part of a richer problem.

Acknowledgements

We are grateful to ILOG SA for providing us with the JSolver toolkit via an Academic Grant licence. We would also like to thank our four reviewers.

References

[1] ILOG JSolver. http://www.ilog.com/products/jsolver/.

[2] D. Gale and L.S. Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 69:9–15, 1962.

[3] I.P. Gent, R.W. Irving, D.F. Manlove, P. Prosser, and B.M. Smith. A constraint programming approach to the stable marriage problem. In *CP'01*, pages 225–239, 2001.

[4] I.P. Gent and P. Prosser. An empirical study of the stable marriage problem with ties and incomplete lists. In *ECAI'02*, 2002.

[5] D. Gusfield and R. W. Irving. *The Stable Marriage Problem: Structure and Algorithms*. The MIT Press, 1989.

[6] Akiko Kato. Complexity of the sex-equal stable marriage problem. *Japan Journal of Industrial and Applied Mathematics (JJIAM)*, 10:1–19, 1993.

[7] A. K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.

[8] C. Unsworth and P. Prosser. A specialised binary constraint for the stable marriage problem. In *SARA05*, 2005.

[9] Pascal van Hentenryck, Yves Deville, and Choh-Man Teng. A generic arc-consistency algorithm and its specializations. *Artificial Intelligence*, 57:291–321, 1992.

Modelling multi-agent systems as constraints for model-based diagnosis

Meir Kalech and **Gal A. Kaminka**
The MAVERICK Group
Computer Science Department
Bar Ilan University
Ramat Gan, Israel
{kalechm, galk}@cs.biu.ac.il

Abstract

With increasing deployment of multi-agent and distributed systems, there is an increasing need for failure diagnosis systems. While successfully tackling key challenges in multi-agent settings, model-based diagnosis has left open the diagnosis of coordination failures, where failures often lie in the boundaries between agents, and thus the inputs to the model—with which the diagnoser simulates the system to detect discrepancies—are not known. However, it is possible to diagnose such failures by modelling the coordination between the agents as constraints. This paper formalizes model-based coordination diagnosis as a constraints satisfaction problem, using two coordination constraints (concurrence and mutual exclusion). The diagnosis process is needed, once some of the constrains are not satisfied. The goal of the diagnosis is to find a minimal set of assignments (by the agents) which violate the satisfiability of the constraints. We define the consistency-based and abductive diagnosis problems within this formalization, and show that both are NP-Hard by mapping them to other known problems. This modelling offers opportunities for cross-cutting research.

1 Introduction

Model-based diagnosis (*MBD*) [Reiter, 1987; de Kleer and Williams, 1987] relies on a model of the diagnosed system, which is utilized to simulate the behavior of the system given the operational context (typically, the system inputs). The resulting simulated behavior (typically, outputs) are compared to the actual behavior to detect discrepancies indicating failures. The model can then be used to pinpoint possible failing components within the system.

MBD is increasingly being applied in distributed and multi-agent systems (e.g., [Fröhlich *et al.*, 1997; Roos *et al.*, 2003; Lamperti and Zanella, 2003]). While successfully addressing key challenges, MBD has been difficult to apply to diagnosing coordination failures [Micalizio *et al.*, 2004]. This is because many such failures take place at the boundaries between the agent and their environment, including other agents. For instance, in a team, an agent may send a message that another agent, due to a broken radio, did not receive. As a result, the two agents come to disagree on an action to be taken. Lacking an omniscient diagnoser that knows of the sending of the message, the receiver has no way to detect and diagnose its fault, since the context—the message that can be fed into a model of the radio of both agents—is unobservable to the diagnoser.

Surprisingly, it is still often possible to detect and diagnose coordination failures, given the actions of agents, and modelling the coordination between the agents as constraints that should ideally be satisfied. In the example above, knowing that the two agents should be under the constraint of agreement as to their actions, and seeing that their actions do not satisfy the constraint (are not in agreement), is sufficient to (1) show that a coordination failure has occurred; and (2) to propose several possible diagnoses for it (e.g., the first agent did not send a message, the second agent did not receive it, etc.).

Some previous works frame the model-based diagnosis problem of a single system as a constraint satisfaction problem [Stumptner and Wotawa, 2003; Sachenbacher and Williams, 2004]. However, they did not address of typical multi-agent systems' failures which take place at the boundaries between the agents and their environment. There are approaches within diagnosis for diagnosing such failures, however, they suffer from key limitations. Fault-based techniques [Horling *et al.*, 2001; Pencolé *et al.*, 2002; Lamperti and Zanella, 2003] utilize pre-enumerated interaction fault models. When the faults are observed, they trigger possible predicted diagnoses. However, the interactions among system entities, in multi-agent system, are not known in advance since they depend on the specific conditions of the environment in runtime and the appropriate actions assigned by the agents [Micalizio *et al.*, 2004]. [Kalech and Kaminka, 2003] propose a technique in which the reasoning of the two agents, leading to their mis-coordinated actions, is re-traced, to determine the roots for their selection. However, this technique is specific to disagreements.

This work takes a first step towards addressing the open challenge of formalizing diagnosis of coordination (*inter-agent*) failures using constraints satisfaction in terms of model-based techniques. We model the coordination between agents as a graph of concurrence and mutual exclusion constraints on agents' actions. The diagnosis process begins with an observation of the agents' actions and inferring, by comparing to the coordination model of the constraints, the minimal number of agents that deviate from the expected coordination (i.e., a minimal set of *abnormal agents*).

The formalization allows definition of both consistency-based and abductive diagnosis problems, and points at several approaches to their solution. While the formalization does not commit to centralized or distributed diagnosis settings, the initial methods we provide are centralized. For consistency-based diagnosis, we show that computing the coordination diagnosis can be mapped to the minimal vertex cover problem. For abductive diagnosis, we take an approach based on constraint satisfaction problem. Both of these problems are thus NP-Hard.

2 Related Work

[Stumptner and Wotawa, 2003; Sachenbacher and Williams, 2004] models the diagnosis problem as a constraints satisfaction problem, and try to use CSP algorithms to compute the diagnosis rapidly. However, they modelled centralized systems where all the components belongs to the same system like boolean circuit. On the other hand, in multi-agent systems beyond the diagnosis to each single agent, the coordination between the agents should be modelled and diagnosed too, in this paper we propose to model the coordination by constraints.

[Pencolé *et al.*, 2002; Lamperti and Zanella, 2003] use a fault-model approach, where the distributed system is modeled as a discrete event system, and the faults are modeled in advance. The diagnoser infers unobservable fault events by computing possible paths in the discrete event system that match observable events. [Horling *et al.*, 2001] and [Micalizio *et al.*, 2004] use causal models of failures and diagnoses to detect and respond to multi-agent and single-agent failures. A common theme in all of these is that they require pre-enumeration of faulty interactions among system entities. However, in multi-agent systems, these are not necessarily known in advance since they depend on the specific run-time conditions of the environment, and the actions taken by the agents.

[Fröhlich *et al.*, 1997], and later [Roos *et al.*, 2003] use a consistency-based approach to diagnose a spatially distributed systems. A set of n agents are responsible for diagnosing n sub-systems, correspondingly. Every agent makes a local diagnosis to its own sub-system and then all agents compute a global diagnosis. In order to build a global diagnosis set, each agent should consider the correctness of those inputs of its subsystem that are determined by other agents. But, Roos et al. and Fröhlich at al. assume that each diagnoser agent knows the context of its sub-system and so it may make the diagnosis. However in our multi-agent system the diagnoser does not have the context so it is impossible to make a diagnosis to every agent separately, unless we supply a model of the coordination between the agents.

[Kalech and Kaminka, 2004] presented a consistency-based diagnosis procedure for behavior-based agents, which utilized a model of behaviors that the agents should be in agreement on (i.e., concurrence coordination). However, their approach was specific only to agreements.

3 Coordinated Multi-Agent Systems

We adopt a model-based approach to diagnosis of coordination failures. To do this, we formalize an agent, a multi-agent system, and the coordination between the agents.

3.1 The Agent Model

An agent is an entity that perceives its environment through sensors and takes actions upon its environment using actuators. Obviously, there are many different possible models that can be used to describe agents. Our focus is on the coordination of multiple agents through their actuators and their sensors, and thus we will use a simplified model, of completely reactive agents, composed only of sensor and actuator components. The connections between the sensors and actuators are described logically.

Definition 1. An *agent* is a pair $\langle CMP, CON\rangle$ of components CMP, and connections CON. CMP is a pair $\langle SEN, ACT\rangle$ where SEN is a set of boolean variables representing the sensors and the ACT is a set of boolean variables representing the actions. CON is a set of logical consequence statements, where the literals of SEN are on the left side of consequences, and the literals of ACT are on the right side.

At any time, the agent may sense through a number of sensors, but may only select one action. Thus multiple literals in SEN may be true, but at any time exactly one literal of ACT must be true. To enforce this, we apply a *completeness* formula (i.e. $ACT_1 \vee \ldots \vee ACT_{|ACT|}$) and a set of *mutual-exclusion* formulas $\forall i, j \neg(ACT_i \wedge ACT_j)$.

Example 1. Suppose we model a scout robot who looks for wounded. The robot has two sensor components, one is a radio sensor with two message values $\{seek, found\}$ and the other is a camera sensor which indicates if the wounded is found. The actions of the robot $\{SEEK, WAIT\}$ are selected based on the sensor readings: Once the robot receives a $seek$ message it selects the action $SEEK$. It will switch to the action $WAIT$ upon finding the wounded (via its camera), or upon receiving a message that it was found (by someone else).

We represent this agent as follows:

$$\begin{array}{rl} SEN = & \{SEN_{radio\_seek}, SEN_{radio\_found}, SEN_{camera\_found}\} \\ ACT = & \{SEEK, WAIT\} \\ CON = & \{SEN_{radio\_seek} \wedge \neg SEN_{camera\_found} \Rightarrow SEEK, \\ & SEN_{radio\_found} \vee SEN_{camera\_found} \Rightarrow WAIT\} \end{array}$$

In addition, we should verify that only one action is selected by the agent, using the following *completeness* and *mutual-exclusion* axioms:

$$WAIT \vee SEEK$$
$$\neg(WAIT \wedge SEEK)$$

3.2 A Model of Coordination

The multi-agent systems of interest to us are composed of several agents, which (by design) are to satisfy certain coordination constraints. We call this type of system a *team*, to distinguish it from general multi-agent systems in which it is possible that no coordination constraints exist.

Definition 2. A *team* T is a set of agents. $T = \{A_1 ... A_n\}$ where A_i is an agent. Given a team T, AS represents the set of the action literals of the agents. Formally, let ACT_i be the set of actions of agent A_i then $AS = \bigcup_{i=1}^{n} ACT_i$, where AS_{ij} represents the j'th boolean action variable of agent A_i. As a shorthand, we use AS_i to denote the boolean action literal of agent A_i whose value is true. We call AS_i the *active selection* of agent A_i.

The actions of agents in a team are coordinated. We utilize two coordination primitives—*concurrence* and *mutual exclusion*—to define the coordination constraints. Concurrence states that two specific actions must be taken jointly, at the same time. Mutual exclusion states the opposite, i.e., that two specific actions may not be taken at the same time.

Definition 3. A *concurrence coordination (CCRN)* constraint between two actions of different agents mandates that the two actions must be true concurrently. Logically, we represent this constraint in a DNF (disjunctive normal form). For two actions AS_{ix} and AS_{jy} (action x of agent A_i and action y of agent A_j) as follows:

$$CCRN(AS_{ix}, AS_{jy}) \Rightarrow (AS_{ix} \wedge AS_{jy}) \vee (\neg AS_{ix} \wedge \neg AS_{jy})$$

Definition 4. A *mutual exclusion coordination MUEX* constraint between two actions of different agents mandates that they cannot be true concurrently. Logically, for two actions AS_{ix} and AS_{jy},

$$\begin{aligned} MUEX(AS_{ix}, AS_{jy}) \quad \Rightarrow \quad & (AS_{ix} \wedge \neg AS_{jy}) \vee \\ & (\neg AS_{ix} \wedge AS_{jy}) \vee \\ & (\neg AS_{ix} \wedge \neg AS_{jy}) \end{aligned}$$

Once we defined the coordination types, we can model the coordination between the agents formally with a set of coordination constraints, defining a graph:

Definition 5. A *coordination graph* for a team T is an undirected graph $CG = \{V, E\}$, where the vertices set V represents the boolean variables of the actions of the agents, and the set of edges E is the set of coordination constraints between the actions. We use CG_m to refer to the m'th constraint within E. $CG(AS_{ix}, AS_{jy})$ denotes the constraint relating AS_{ix} and AS_{jy}. CG_m is considered true if the constraint holds and false otherwise.

Example 2. Figure 1 presents a coordination graph. The concurrence constraints are represented by solid lines, and the mutual exclusion constraints are represented by dashed lines. Assume a team of three agents $\{A_1, A_2, A_3\}$. A_1 and A_2 are scout robots as described in Example 1, and A_3 is a paramedic robot who has one radio sensor with one message value *{found_message}*, and three actions *{JOIN, TREAT, CHARGE}*. Agents A_1 and A_2 have the same role in the team so they have *concurrence coordination* constraints between their actions. At the beginning A_1 and A_2 receive a $seek$ message so they select the action $SEEK$ while A_3 may select any action except $TREAT$, meaning it can not treat a wounded, while the other robots seek. We can see the *mutual exclusion coordination* constraints between these behaviors. Once A_1 or A_2 find the wounded, they send a $found\_message$ to the other agents in the team, then A_1 and A_2 transport to the $WAIT$ action, while A_3 transports to $JOIN$ action. Again we can see the *concurrence coordination* constrains between these behaviors. In addition, when agent A_3 is being charged ($CHARGE$ behavior), there are no constraints between the agents. The corresponding CG is formally defined as follows:

$$\begin{aligned} V = \; & \{AS_{1_{WAIT}}, AS_{2_{WAIT}}, AS_{1_{SEEK}}, AS_{2_{SEEK}}, \\ & AS_{3_{TREAT}}, AS_{3_{JOIN}}, AS_{3_{CHARGE}}\} \\ E = \; & \{CCRN(AS_{1_{WAIT}}, AS_{2_{WAIT}}), \\ & CCRN(AS_{1_{SEEK}}, AS_{2_{SEEK}}), \\ & MUEX(AS_{2_{SEEK}}, AS_{3_{TREAT}}), \\ & CCRN(AS_{2_{WAIT}}, AS_{3_{JOIN}}), \\ & MUEX(AS_{1_{SEEK}}, AS_{3_{TREAT}}), \\ & CCRN(AS_{1_{WAIT}}, AS_{3_{JOIN}})\} \end{aligned}$$

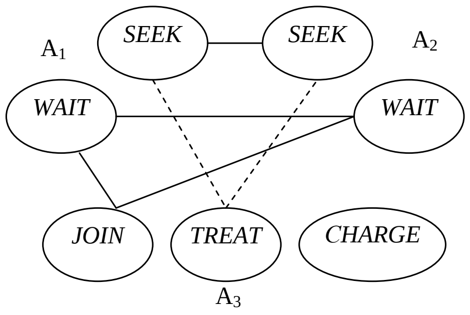


Figure 1: **The coordination graph for team $\{A_1, A_2, A_3\}$.**

Given a coordination graph CG and a team T, we can define a multi-agent system description as a set of implications from the normality of the agents to the satisfaction of the coordination constraints. This is the final piece in formalizing a normally-functioning multi-agent system.

Definition 6. A *multi agent system description (MASD)* is a set of implications from the normality of agents in a team T to CG. The meaning of the predicate $AB(\cdot)$ is that the corresponding agent is considered abnormal (failing).

$$\begin{aligned} MASD = \; & \{\neg AB(A_i) \wedge \neg AB(A_j) \Rightarrow CG(AS_{ix}, AS_{jy}) \\ & |CG(AS_{ix}, AS_{jy}) \in CG \wedge A_i, A_j \in T\} \end{aligned}$$

4 Diagnosis of Coordination Faults

A violation of the coordination constraints may be the result of a fault in one of the sensors or other agent components [1] Given an $MASD$ and a set of normality assumptions, it is possible to infer that a fault exists (and to generate hypotheses as to its identity), by checking whether the observed actions of the agents satisfy the $MASD$.

[1]It may also be the result of a fault in the environment, e.g., when a message is lost in transit. This is treated as a fault in the receiver.

Let us formalize the coordination diagnosis in terms of model based diagnosis:

Definition 7 *Coordination Diagnosis Problem***.** Given $\{T, MASD, AS\}$ where T is a team of agents $\{A_1...A_n\}$, $MASD$ is a multi agent system description defined over T (Def. 6), and AS is the set of the actions of the agents (Def. 2), then the *coordination diagnosis problem (CDP)* arises when

$$MASD \cup \{\neg AB(A_i)|A_i \in T\} \cup AS \vdash \bot$$

We use the following example to illustrate.

Example 3. Suppose we are given the following $MASD, T$, and AS (only the true literals in AS are shown):

$$\begin{aligned}
T = & \quad \{A_1, A_2, A_3, A_4, A_5, A_6\}\\
MASD = & \quad \{\neg AB(A_1) \wedge \neg AB(A_4) \Rightarrow MUEX(AS_{11}, AS_{41}),\\
& \neg AB(A_1) \wedge \neg AB(A_2) \Rightarrow CCRN(AS_{12}, AS_{21}),\\
& \neg AB(A_1) \wedge \neg AB(A_6) \Rightarrow CCRN(AS_{12}, AS_{61}),\\
& \neg AB(A_2) \wedge \neg AB(A_3) \Rightarrow CCRN(AS_{22}, AS_{31}),\\
& \neg AB(A_2) \wedge \neg AB(A_5) \Rightarrow CCRN(AS_{22}, AS_{51}),\\
& \neg AB(A_2) \wedge \neg AB(A_6) \Rightarrow CCRN(AS_{21}, AS_{61}),\\
& \neg AB(A_3) \wedge \neg AB(A_4) \Rightarrow MUEX(AS_{32}, AS_{42}),\\
& \neg AB(A_3) \wedge \neg AB(A_5) \Rightarrow CCRN(AS_{31}, AS_{51})\}\\
AS = & \quad \{AS_{11}, AS_{21}, AS_{31}, AS_{41}, AS_{51}, AS_{61}\}
\end{aligned}$$

Figure 2 shows the coordination graph for this CDP. Assuming all the agents are not abnormal, the actions of certain agents violate the constraints satisfaction and so they are not consistent with the coordination graph. For instance, the actions $AS_{11} = true$ and $AS_{41} = true$ causes an inconsistency in CG_1, as it produces a false value of $MUEX(AS_{11}, AS_{41})$, though, $MUEX(AS_{11}, AS_{41})$ should be true, given the normality assumptions $\neg AB(A_1), \neg AB(A_4)$. On the other hand, if the actions $AS_{12}, AS_{21}, AS_{32}, AS_{41}, AS_{52}, AS_{61}$ were true (implying that the other actions were false), they would have been consistent with the coordination graph.

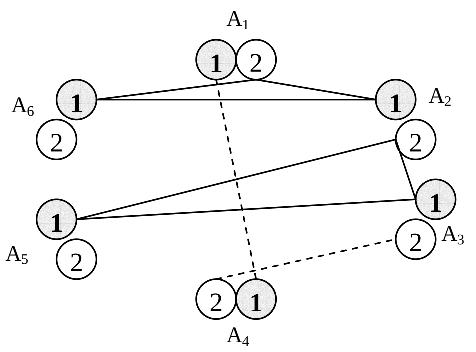


Figure 2: **The coordination graph and active selection (gray circles) of the team** $T = \{A_1, A_2, A_3, A_4, A_5, A_6\}$

Given a CDP, the goal of the coordination diagnosis process is to determine a minimal set of abnormal agents whose selection and subsequent setting of the $AB(\cdot)$ clause would eliminate the inconsistency (consistency-based diagnosis, Section 4.1), or explain it (abductive diagnosis, Section 4.2). In terms of CSP, the set of abnormal agents would explain the constraints satisfaction violation. A coordination diagnosis (a set of abnormal agents) is minimal, iff no proper subset of it is a coordination diagnosis.

Once the set of such abnormal agents is found, the diagnoser infers the abnormal components (in our case, sensors) within the abnormal agents. This is done using straightforward back-chaining through the set CON (Def. 1) of logical consequence statements connecting sensors to actions (e.g., as in [Kalech and Kaminka, 2003]).

4.1 Consistency-Based Coordination Diagnosis

We begin by defining consistency-based coordination diagnosis.

Definition 8. A *consistency-based global coordination diagnosis (CGCD)* is a minimal set $\Delta \subseteq T$ such that:

$$MASD \bigcup \{AB(A_i)|A_i \in \Delta\} \bigcup \{\neg AB(A_i)|A_i \in T - \Delta\} \bigcup AS \nvdash \bot$$

The first step in this process to determine which agents are in conflict (violate the constraints between them):

Definition 9. Two agents a and b are called *conflict pair* $\langle a, b\rangle$, if there exist a constraint CG_i that relates a and b and whose value is false.

$$\forall a, b \in T, \exists i, j, k \; s.t. \neg CG_i(AS_{aj}, AS_{bk}) \Rightarrow \langle a, b\rangle$$

Definition 10. A *local conflict set* is a set of the all conflict pairs in the system, and is denoted by LC.

Example 4. LC in the graph of example 3 is: $LC = \{\langle A_1, A_4\rangle \langle A_1, A_2\rangle, \langle A_1, A_6\rangle, \langle A_2, A_3\rangle, \langle A_2, A_5\rangle\}$

The local conflict set forms the basis for the $CGCD$, because for each conflict pair, at least one of the agents is abnormal. However, the $CGCD$ is not a simple combination of all agents in the LC pairs, as arbitrary selection of agents may lead to diagnosis sets that are themselves inconsistent. For instance, treating each pair in the computed LC in Example 4 by itself, produces the following subset of possible diagnoses:

$$\begin{aligned}
\langle A_1, A_2\rangle &\Rightarrow \{AB(A_1), \neg AB(A_2)\}\\
\langle A_1, A_2\rangle &\Rightarrow \{\neg AB(A_1), AB(A_2)\}\\
\langle A_1, A_4\rangle &\Rightarrow \{AB(A_1), \neg AB(A_4)\}\\
\langle A_1, A_4\rangle &\Rightarrow \{\neg AB(A_1), AB(A_4)\}
\end{aligned}$$

It is easy to see that combining these diagnoses may produce inconsistency (for instance, combining the first and last implications would produce the set $\{AB(A_1), \neg AB(A_2), \neg AB(A_1), AB(A_4)\}$).

Therefore, we cannot diagnose every conflict pair by itself and then combine the results. Rather, we should compute the diagnoses sets Δ considering the dependencies between the conflict pairs. To do this, we should look for the abnormal agent(s) in every conflict pair.

We achieve this goal by generating a hitting-set of agents, selecting at least one agent as abnormal from every conflict pair, such that the resulting agents cover between them all conflict pairs. We want to maintain a minimal number of such agents. This is somewhat similar to Reiter's HS-Tree [1987], or de Kleer and Williams' technique [1987]. It is also related to minimal model techniques used in non-monotonic reasoning [Olivetti, 1992; Niemelä, 1996]. We plan to explore these connections in the future.

We achieve this goal by transforming the conflict set into a graph, and finding the vertex cover for this graph. Let us define a conflict graph $G = \{V', E'\}$ where E' is a set of the conflict pairs and V' is a set of the agents involved in the conflict set. In order to compute the diagnosis we run an algorithm to find a minimal vertex cover—a set of vertices that involve all edges. A vertex cover set is guaranteed to be a diagnosis since all the edges, namely the conflict pairs, are covered by this set, namely by a set of abnormal agents. We are looking for all the possible minimal vertex cover sets, since the diagnosis contains all the possibilities of abnormal agents. Minimal vertex covers guarantee minimal diagnosis, since a vertex cover is minimal only if no proper subset of it is a vertex cover.

Determining a minimal vertex cover is known to be NP-Complete, however the problem of determining the set of minimal vertex covers is NP-Hard [Skiena, 1990]. A simple $O(2^{|V|})$ exact algorithm for its solution is to find all the possible vertex covers in size one, then continue to find the possible vertex covers in size two, under the condition that it is not a superset of a previous vertex cover, and so on up to the max size of the graph. The complexity of computing the $CGCD$ is thus the same as in single-agent diagnosis methods, e.g., [de Kleer and Williams, 1987].

Example 5. Figure 3 presents the graph of the conflict pairs that were computed in example 4. The vertex cover set of size one is empty, for size two it is $VC_1 = \{A_1, A_2\}$, and there are two sets of size three: $VC_2 = \{A_1, A_3, A_5\}$ and $VC_3 = \{A_2, A_4, A_6\}$ (there are more vertex cover sets which are superset of VC_1), it is unnecessary to continue to check the vertex cover in size four and more since every such vertex cover will be a superset of the formers. By building the vertex cover sets we obtain the global coordination diagnosis, $\Delta_1 = \{A_1, A_2\}$, $\Delta_2 = \{A_1, A_3, A_5\}$, $\Delta_3 = \{A_2, A_4, A_6\}\}$.

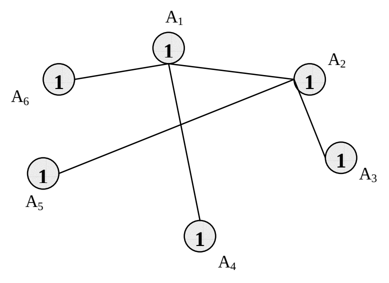


Figure 3: **A graph of the conflict pairs in example 4**

A disadvantage of the consistency-based approach is that it may produce diagnoses that are unsound, in the sense that while they eliminate the inconsistency, they do not explain it. Intuitively, such diagnoses correspond to eliminating the abnormal agents from consideration, rather than suggesting that they change their actions. For such diagnoses, there may be no actions that the abnormal agents could take in such a way that the constraints in $MASD$ will be satisfied.

For instance, in Example 5 the diagnosis set $\{A_1, A_2\}$ represents a minimal set of abnormal agents, but changing their actions $(A_{11} = false, A_{12} = true, A_{21} = false, A_{22} = true)$ will leave the constraints system unsatisfied with $CCRN(AS_{12}, AS_{21}) = false$. On the other hand, changing the actions of the agents in the other diagnoses ($\{A_1, A_3, A_5\}, \{A_2, A_4, A_6\}$) will eliminate the inconsistency.

4.2 Abductive Coordination Diagnosis

The implication is that stronger conditions on the solution sets may be needed. Such conditions correspond to abductive diagnosis, in which changing the actions of the abnormal agents entails the coordination graph:

Definition 11. An *abductive global coordination diagnosis (AGCD)* is a minimal set $\Delta \subseteq T$ such that:

$$MASD \bigcup \{AB(A_i)|A_i \in \Delta\} \bigcup \{\neg AB(A_i)|A_i \in T - \Delta\} \bigcup AS \nvdash \bot$$

and,

$$\{AB(A_i)|A_i \in \Delta\} \bigcup \{\neg AB(A_i)|A_i \in T - \Delta\} \bigcup AS \Rightarrow CG$$

where, we make the active selection of agent A_i (Def. 2), AS_i, false, and force A_i to choose a different action,

$$AB(A_i) \Rightarrow \neg AS_i \wedge (AS_{i1} \vee \ldots \vee AS_{i|ACT|})$$

The first condition in Def. 11 is exactly as in Definition 8 (i.e., $CGCD$) to satisfy the consistency requirement. The second condition requires that for any abnormal agents found, it will be possible to change their active selection, in order to entail the coordination graph and thus satisfy the coordination constraints. Note that the entailment here is of the coordination graph, not the full $MASD$.

The unsound diagnosis set $\{A_1, A_2\}$, given by the consistency-based approach (in Example 5), will not pass this second condition, since the alternative actions of agent A_1 and of agent A_2 do not entail the coordination graph.

In order to satisfy Definition 11, the diagnosis process needs to go beyond pinpointing suspect agents, to verifying that by changing their actions, coordination will be restored. Thus in contrast with consistency-based approach, we do not utilize conflict pairs to compute the diagnoses, but instead examine all action literals assignments that entail the coordination graph, i.e., all actions which will satisfy the coordination constraints. Then the process compares the existing truth values to those that will satisfy the coordination, and computes a *minimal* set of changes.

Example 6. Let us compute the $AGCD$ of the Example 3. Table 1 presents the satisfying truth assignments for the actions of agents $A_1 \ldots A_6$. There are only two such assignments. In order to find the minimal *AGCD*, we should compare the actions of the agents with these assignments and point out the agents that deviate. Consider the actions in

Example 3 (where $AS_{11}, AS_{21}, AS_{31}, AS_{41}, AS_{51}, AS_{61}$ are true, and the other action literals are false). Then, in the first row $AS_{11} = false$, but we have $AS_{11} = true$. We thus mark action AS_{11} as faulty. The second value in the table is $AS_{12} = true$, but we have $AS_{12} = false$, so we again mark this as faulty, and so on for each one of the actions. For the first entry in the truth table we got the following faulty actions: $AS_{11}, AS_{12}, AS_{31}, AS_{32}, AS_{51}, AS_{52}$. From this list, we can determine the abnormal agents by finding the agents whose actions are faulty. We thus conclude that a minimal *AGCD* is $\Delta_1 = \{A_1, A_3, A_5\}$ for this row. From the second row, we similarly find $\Delta_2 = \{A_2, A_4, A_6\}$. Setting these agents to abnormal, and thus forcing them to select different actions, would satisfy the coordination constraints.

#	A_1		A_2		A_3		A_4		A_5		A_6	
	1	**2**	**1**	**2**	**1**	**2**	**1**	**2**	**1**	**2**	**1**	**2**
1	0	1	1	0	0	1	1	0	0	1	1	0
2	1	0	0	1	1	0	0	1	1	0	0	1

Table 1: **Coordination-satisfying actions in Example 6.**

Obviously, we should consider only the minimal *AGCD*. We fulfill this requirement by comparing every new hypothesized coordination diagnosis to the former coordination diagnoses, and checking whether it is a subset, a superset, or different than the former diagnoses.

Thus the $AGCD$ problem is essentially that of finding all sets of truth assignments that will satisfy a target proposition, an NP-Hard problem. A detailed discussion of satisfiability, and the rich literature offering efficient exact and approximate solution methods is well beyond the scope of this paper. However, we point at two diagnosis-specific mechanisms that can potentially be used to alleviate computational load in our case:

1. Ordered binary decision diagram (OBDD) [Bryant, 1992] can be used to efficiently reason about diagnosis-satisfying assignments [Torasso and Torta, 2003]. By restricting the representation, boolean manipulation becomes much simpler computationally. We can compactly represent the coordination graph using OBDDs (an off-line construction process), and then truth assignments can be computed in linear time in many cases.

2. Assumption-based truth maintenance systems (ATMS) [de Kleer, 1986] can be used to build the satisfying assignments incrementally. We exploit the fact that it is unnecessary to check all the assignments since the legal assignments depend each on the other. For instance, assume a concurrence coordination between a and b and between b and c:

$$\begin{array}{ll} & ((a \wedge b) \vee (\neg a \wedge \neg b)) \\ \bigwedge & ((b \wedge c) \vee (\neg b \wedge \neg c)) \end{array}$$

Instead of computing the full truth table of a, b and c, (2^3), we can use an ATMS, which given these justifications will provide only two assignments: $(a = true, b = true, c = true)$ or $(a = false, b = false, c = false)$.

5 Summary and Future Work

We presented a novel formalization for diagnosing coordination failures in multi agent systems by representing the coordination as constraints between agents which must be satisfied. We model such coordination using two coordination constraints (concurrence and mutual exclusion). In the diagnosis process the diagnoser observes the actions of the agents, then it finds the candidate abnormal agents who violated the constraints by the coordination model, and finally continues to compute the abnormal sensors by back-chaining (previously shown in [Kalech and Kaminka, 2003]).

We defined both a consistency-based and abductive diagnosis versions of coordination diagnosis, and proposed initial algorithms for both. The consistency-based approach finds the local conflicts between pairs of agents, then continues to compute the diagnosis by combining the conflicts using a minimal vertex cover algorithm. We showed that this approach is unsound, in that it may produce diagnoses that are impossible since any transformation of their value will not satisfy the constraints between the agents. The second approach maps the abductive coordination diagnosis problem to that of satisfiability, finding a minimal set of truth-value changes that satisfy a given proposition. Here, our initial approach pre-computes all the possible coordination-satisfying action assignments, and then uses these during on-line diagnosis by comparing the actions of the agents to each one of the instances of the satisfying action assignments.

Our goal in this paper was to take a first step towards the use of model-based diagnosis techniques in multi-agent systems, by representing it as CSP. Naturally, much is left for future research. First, the algorithms we proposed are related to key techniques in diagnosis, constraint-satisfaction, and non-monotonic reasoning. We plan to explore these connections, to bring to bear on this diagnosis problem. Second, we intentionally avoided the use of complex multi-component agent models, and focused on simple coordination primitives. We hope to explore richer models of both in the future. In addition, while this paper has adopted the perspective of a centralized single diagnoser, we plan to tackle distributed algorithms next. Representing the model-based diagnosis of coordination as CSP opens new opportunity to inspire both of the areas: distributed CSP as well as diagnosis of multi-agent systems.

Acknowledgments

This research was supported in part by BSF grant 2002401. We gratefully acknowledge a generous gift from IBM. As always, thanks to K. Ushi and K. Ravit.

References

[Bryant, 1992] Randal E. Bryant. Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.

[de Kleer and Williams, 1987] J. de Kleer and B. C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32(1):97–130, 1987.

[de Kleer, 1986] Johan de Kleer. An assumption-based truth maintenance system. *Artificial Intelligence*, 28:127–162, 1986.

[Fröhlich *et al.*, 1997] Peter Fröhlich, Iara de Almeida Mora, Wolfgang Nejdl, and Michael Schröder. Diagnostic agents for distributed systems. In *ModelAge Workshop*, pages 173–186, 1997.

[Horling *et al.*, 2001] Bryan Horling, Brett Benyo, and Victor Lesser. Using Self-Diagnosis to Adapt Organizational Structures. *Proceedings of the 5th International Conference on Autonomous Agents*, pages 529–536, June 2001.

[Kalech and Kaminka, 2003] Meir Kalech and Gal A. Kaminka. On the design of social diagnosis algorithms for multi-agent teams. *in International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 370–375, 2003.

[Kalech and Kaminka, 2004] Meir Kalech and Gal A. Kaminka. Diagnosing a team of agents: Scaling-up. *In Proceedings of the International Workshop on Principles of Diagnosis (DX 2004)*, pages 129–134, 2004.

[Lamperti and Zanella, 2003] Gianfranco Lamperti and Marina Zanella. *Diagnosis of Active Systems*. Kluwer Academic Publishers, 2003.

[Micalizio *et al.*, 2004] R. Micalizio, P. Torasso, and G. Torta. On-line monitoring and diagnosis of multi-agent systems: a model based approach. *in Proceeding of European Conference on Artificial Intelligence (ECAI 2004)*, 16:848–852, 2004.

[Niemelä, 1996] I. Niemelä. A tableau calculus for minimal model reasoning. In *Proceedings of the fifth workshop on theorem proving with analytic tableaux and related methods*, 1996.

[Olivetti, 1992] N. Olivetti. A tableaux and sequent calculus for minimal model entailment. *Journal of automated reasoning*, 9:99–139, 1992.

[Pencolé *et al.*, 2002] Y. Pencolé, M.O. Cordier, and L. Rozé. Incremental decentralized diagnosis approach for the supervision of a telecommunication network. *IEEE Conference on Decision and Control (CDC'02)*, pages 435–440, December 2002.

[Reiter, 1987] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–96, 1987.

[Roos *et al.*, 2003] Nico Roos, Annette ten Teije, and Cees Witteveen. A protocol for multi-agent diagnosis with spatially distributed knowledge. *in Proceedings of Autonomous Agents and Multi Agent Systems (AAMAS-03)*, pages 655–661, July 2003.

[Sachenbacher and Williams, 2004] Martin Sachenbacher and Brian Williams. Diagnosis as semiring-based constraint optimization. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI-2004)*, 2004.

[Skiena, 1990] Steven Skiena. *Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica*. Addison-Wesley, 1990.

[Stumptner and Wotawa, 2003] Markus Stumptner and Franz Wotawa. Coupling csp decomposition methods and diagnosis algorithms for tree-structured systems. In *In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2003.

[Torasso and Torta, 2003] Pietro Torasso and Gianluca Torta. Computing minimum-cardinality diagnoses using OBDDs. *Advances in Artificial Intelligence (lecture notes in artificial intelligence)*, 2281:224–238, 2003.

A Constraint-Based Planner for Data Production

Wanlin Pang[1] **Keith Golden**
NASA Ames Research Center
Moffett Field, CA 94035
{wpang, kgolden}@email.arc.nasa.gov

Abstract

This paper presents a graph-based backtracking algorithm designed to support constraint-based planning in data production domains. This algorithm performs backtracking at two nested levels: *outer-backtracking* following the structure of the planning graph to select planner subgoals and actions to achieve them and *inner-backtracking* inside a sub-problem associated with a selected action to find action parameter values. We show that this algorithm works well in a planner applied to automating data production in an ecological forecasting system. We also discuss how the idea of multi-level backtracking may improve the efficiency of solving semi-structured constraint problems.

1 Introduction

Earth-science data processing (ESDP) at NASA is a data production problem of transforming low-level observations of the Earth system, such as remote sensing data, into high-level observations or predictions, such as crop failure or high fire risk. Given the large number of socially and economically important variables that can be derived from the data, the complexity of the data processing needed to derive them and the many terabytes of data that must be processed each day, there are great challenges and opportunities in processing the data in a timely manner, and a need for more effective automation. Our approach to providing this automation is to cast it as a planing problem: we represent data-processing operations as planner actions and desired data products as planner goals, and use a planner to generate data-flow programs that produce the requested data products.

Many of the recent advances in planning, such as state-based heuristic search or reduction to satisfiability problems, are not readily adapted to ESDP problems, due to some of its particular features, such as incomplete information, large and dynamic universes, complex data types, and complex constraints, just to name a few.

[1]QSS Group Inc

We take the approach, like many other researchers [van Beek & Chen, 1999; Lopez & Bacchus, 2003; Do & Kambhampati, 2001; Smith, Frank, & Jónsson, 2000], of translating the planning problem into a constraint satisfaction problem (CSP). However, since data processing domains are substantially different from other planning domains that have been explored, our approach to translating planning problems to CSPs differs as well. For example, [Do & Kambhampati, 2001] use variables to represent goals and domains to represent available planner actions achieving the goals. Constraints are used to encode mutual exclusion relations. While this is an effective approach for propositional planning problems, we also need variables to represent objects and action parameters, and constraints to represent relations among them. Thus, our encoding is somewhat more complex, and the CSPs resulting from our encoding are hard to solve by the search methods employed in other planners [van Beek & Chen, 1999; Lopez & Bacchus, 2003; Do & Kambhampati, 2001; Smith, Frank, & Jónsson, 2000].

We have developed a constraint-based planner, called DoPPLER, for **d**ata **p**rocessing **planner**. From a data processing task, the planner constructs a *lifted planning graph,* from which it derives a CSP representation of the planning problem and then searches the CSP for a solution. Whereas a conventional planning graph [Blum & Furst, 1997] is a grounded representation, consisting of ground actions and propositions, a *lifted* planning graph contains variables. This is not only a much more concise representation than an ordinary planning graph, but it also is the only practical way that we know to represent potentially infinite sets of ground actions. Even though the CSP derived from a lifted planning graph is difficult to solve by many existing CSP search methods such as chronological backtracking (BT), forward checking (FC) or conflict-directed backjumping (CBJ), it has certain structural properties inherited from the planning graph. We have developed a search algorithm based the structure of the planning graph to improve the efficiency of solving the CSP.

In this paper, we report on our work applying DoPPLER to automating data production problems. We discuss how the data production problem is cast as a planning problem which, in turn, is translated into a CSP, and how the planning graph is used to improve backtracking in solving the CSP. Section 2 discusses data processing as a planning task and our planning approach. Section 3 describes the graph-based planning search algorithm. Section 4 presents the preliminary testing results, and Section 5 discusses related and future work.

2 Planning for Data Processing

Data processing is a task of transforming data products into other data products. A common sequence of data processing steps is: 1) gather data from multiple sources; 2) convert the data into a common representation; 3) combine the data and perform other transformations; 4) feed the data into science models and then run the models; 5) convert the output of the model into some form suitable for visualization; 6) repeat some or all these steps depending on the requirements. To formalize data processing as a planning problem, we represent data-processing operations as planner actions, desired data products as planner goals, and available data sources as part of the initial state.

Planning in DoPPLER is a two-stage process. The first stage consists of a Graphplan-style reachability analysis [Blum & Furst, 1997] to derive heuristic distance estimates for the second stage, a constraint-based search. These stages are not entirely separate, however; constraint propagation occurs in both graph-construction and constraint search stages, and the graph is refined during the constraint search phase.

2.1 Actions and Conditions

An action is a tuple $\langle \mathcal{I}, \mathcal{O}, \mathcal{P}, \Pi, \mathcal{E}, \chi \rangle$, where $\mathcal{I}$,$\mathcal{O}$,$\mathcal{P}$ are the *input* variables, *output* variables and *parameters*, respectively. All these variables are typed. Π is the precondition, $\mathcal{E}$ is a list of *effects* and χ is a procedure for executing the action that may reference any variable in $\mathcal{I} \cup \mathcal{P}$ and must set every variable in $\mathcal{O}$.

A full discussion of preconditions and effects in the **D**ata **P**rocessing **A**ction **D**escription **L**anguage (DPADL) can be found in [Golden, 2002]; for the purposes of this paper, it suffices to observe that many goals and preconditions consist of requirements on the attributes of variables in $\mathcal{I}$ and many effect conditions consist of assignments to the attributes of variables in $\mathcal{O}$ and creation of new objects (which themselves are specified in terms of assignments on attributes). These conditions can be expressed in the concise canonical form $v = \langle a_1, a_2, \ldots, a_n \rangle$, where v is a variable and $\langle a_1, a_2, \ldots, a_n \rangle$ is a structure specification, where each attribute a_i may be a variable, constant, structure specification, or $\emptyset$ (unspecified). For example, suppose the attributes of a file are name, format, and size. A file can be represented as a tuple⟨name, format, size⟩. To specify the goal of finding a file f named "foo.txt" whose size is greater than 100, we could write $f = \langle \texttt{"foo.txt"}, \emptyset, s \rangle \wedge s > 100$.

Like goals, effects can also have $\emptyset$-attributes, but the meaning is different. In goals, $\emptyset$ means *don't care.* In effects, it means *default.* Any variable $o \in \mathcal{O}$ or new object may be specified as a *copy* of some variable $d \in \mathcal{I}$, in which case attributes of o default to the same value as attributes of d. If nothing else were specified, then o would be a perfect copy of d. However, what we are interested in is typically not perfect copies, but imperfect ones. For example, there are many actions that change just one or two properties of an object, such as file format, projection, resolution, size, or name. Specifying the outputs of those actions as copies of their inputs allows us to list only the attributes that are changed [Golden, 2003]. In our canonical form, an effect that changed only one attribute of o would be of the form $o = \langle \emptyset, \emptyset, \ldots, n, \ldots \emptyset, \emptyset \rangle$, where n is the new value for the attribute that changed. All other attributes take on the corresponding value from d.

2.2 Lifted Planning Graphs

From the planning problem specification, the planner incrementally constructs a directed graph, similar to a planning graph [Blum & Furst, 1997], but using a lifted representation (*i.e.*, containing variables). Arcs in the graph are analogous to causal links [Penberthy & Weld, 1992]. A causal link is triple $\langle \alpha_s, p, \alpha_p \rangle$, recording the decision to use action α_s to support precondition p of action α_p. However, instead of an arc to record a commitment of support, we use it to indicate the ***possibility*** that α_s supports p. The lifted graph contains multiple ways of supporting p; the choice of the actual supporter is left to constraint search. We add an extra term to the arc for bookkeeping purposes – the condition $\gamma_p^{\alpha_s}$ needed in order for α_s to achieve p. A link then becomes $\langle \alpha_s, \gamma_p^{\alpha_s}, p, \alpha_p \rangle$.

Given an unsupported precondition p of action α_p, our first task is to identify all the actions that could support p. Because the universe is large and dynamic, identifying all possible ground actions that could support p would be impractical, so instead we use a lifted representation, identifying all action *schemas* that could provide support. Given an action schema α, we determine whether it supports p by ***regressing*** p through α_s. The result of regression is the formula $\gamma_p^{\alpha_s}$. If $\gamma_p^{\alpha_s} = \perp$, then α_s does not support p. Initial graph construction starts from the planner goal and terminates when all preconditions have support or (more likely) a potential loop is detected. Section 2.5 discusses the planning graph construction in more detail.

2.3 From Planning to Constraints

After the graph is constructed, heuristic distance estimates for guiding the search are computed, and a constraint network representing the search space is incrementally built. It is incremental because the planning graph comprises a compact representation of the search space, in which each action node can represent multiple concrete actions in the final plan. Since the number of possible actions can be large, even infinite, we cannot simply generate all of them at once but do so lazily during search. This is handled using a dynamic CSP

(DCSP), in which new variables and constraints can be added for each new action and causal link in the plan.

However, it is not always necessary to ground out all of the actions in the CSP. As discussed in [Golden & Frank, 2002], our constraint reasoning techniques can handle constraints that include universally quantified variables, and we use this to our advantage when solving universally quantified goals or preconditions in domains that are highly symmetric. For example, consider the subgoal of constructing a mosaic, using the LAZEA projection, from all tiles covering the continental US for a given day and satellite data product. A precondition of the `mosaic` action will be that the input tiles are all in the LAZEA projection. This precondition can be satisfied for any given tile by using the `reproject` action.

Just as one node in the graph can represent multiple concrete actions, one concrete action can be represented by multiple nodes. That is, two nodes in the graph might actually *unify*. We don't commit at planning time to whether a condition is supported by a new or existing action, and having two separate nodes for a given action schema (or two action variables in the constraint network) does not necessarily mean that there are two distinct instances of that schema in the plan. Similarly, two object variables may both designate the same object. The burden this least commitment approach imposes on the constraint network is an additional $O(n^2)$ constraints for every set of n variables that could conceivably unify. For example, suppose we have two action variables a_1 and a_2, both representing instances of `reproject`, which has one output. We will represent the output variables of $a_1$and a_2 as $a_1.out$ and $a_2.out$, respectively. Since two distinct actions cannot have the same output, if both outputs variables are forced to codesignate (for example, because each is constrained to be the sole input of a single concrete action), then the action variables must also codesignate: $(a_1.out = a_2.out) \Rightarrow (a_1 = a_2)$. Similarly, if the actions codesignate, then their corresponding inputs and parameters must also codesignate. We are exploring an alternative representation of these constraints that avoids explicitly generating all $O(n^2)$ constraints.

The constraints generated for a given planning problem are simply a naive translation of explanatory frame axioms corresponding to the planning problem. We have boolean variables for all of the arcs (causal links) and conditions in the plan. For each condition c, we have a constraint specifying that exactly one of the possible causal links l_i supporting that condition is chosen:

$$\text{ImpliesXOR}(c, l_1, \ldots, l_n),\ \textit{i.e.},\ c \Rightarrow l_1 \otimes \ldots \otimes l_n$$

For each link l and each condition c_k that the link can support (a link can support multiple conditions when each condition is an attribute of an immutable object), we have a constraint stating that if l is chosen, then c_k is true iff a condition $cn(l, c_k)$, obtained by regressing c_k through the action, is true.

$$\text{ImpliesEqual}(l, cn(l, c_k), c_k){:} l \Rightarrow cn(l, c_k) = c_k$$

Conditions such as $cn(l, c_k)$, which is obtained by goal regression, may correspond to fairly complex expressions. These are represented in a very straightforward manner. For example, given the expression $x = y \vee (x = 1 \wedge y = 2)$, we introduce new boolean variables v_{or}, v_{and}, v_{eq}, v_1, v_2, and the following constraints

$$\begin{aligned}
&\text{CondOr}(v_{or}, v_{eq}, v_{and}){:}\ v_{or} \Leftrightarrow v_{eq} \vee v_{and}\\
&\text{CondAnd}(v_{and}, v_1, v_2){:}\ v_{and} \Leftrightarrow v_1 \wedge v_2\\
&\text{CondEqual}(v_{eq}, x, y){:}\ v_{eq} \Leftrightarrow x = y\\
&\text{CondEqual}(v_1, x, 1){:}\ v_1 \Leftrightarrow x = 1\\
&\text{CondEqual}(v_2, y, 2){:}\ v_2 \Leftrightarrow y = 2
\end{aligned}$$

In summary, the constraint problem derived from the planning graph contains: 1) boolean variables for all arcs, nodes and conditions; 2) variables for all parameters, input and output variables and function values; 3) for every condition in the graph, a constraint specifying when that condition holds (for conditions supported by arcs, this is just the XOR of the arc variables); 4) for conjunctive and disjunctive expressions, the constraint is the respective conjunction or disjunction of the boolean variables corresponding to appropriate sub-expressions; 5) for every arc in the graph, constraints specifying the conditions under which the supported fluents will be achieved (i.e., $\gamma_p^\alpha \Rightarrow p$, where γ_p^α is the precondition of α needed to achieve p) ; 6) user-specified constraints; and 7) constraints representing structured objects.

2.4 Planning Search

Guided by heuristic distance estimates extracted from the planning graph, the planner first selects planner subgoals to achieve and actions to achieve them, which form a *lifted plan*. After the subgoal and action selection, the CSP solver finds values for variables representing planner action parameters. This is necessary to make actions executable. During the search, propagation is performed whenever a value is assigned to a variable. The search is an iterative process involving possible backtracks; that is, if there are no valid parameters for a chosen action, the planner has to search for another plan; if it is impossible to extract a plan from the current planning graph, the planning graph has to be extended, or the planner admits the failure of finding a plan.

2.5 A Simplified Example

A typical data processing task consists of gathering data files, transforming them, feeding them into a science model (e.g., a fire-risk model), and producing a final data file so that a decision maker can assess the fire risk of a particular region. For simplicity, we ignore much of the complexity of the data processing domain and focus on one sub-problem: spatial aggregation. So a simplified task becomes to take some regions from thousands of available regions and compose them to create a mosaic that covers a specified region.

Specifically, a *region* is a pair of points $\langle ul, lr \rangle$ where ul is the upper-left corner and lr the lower-right corner. A point is a pair of coordinates (x, y). Normally x and y would be longitude and latitude, but as a further simplification, we will assume both x and y are non-negative

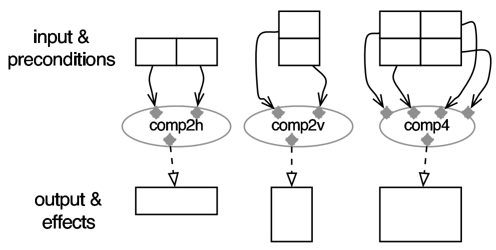


Figure 1: The planner actions : the dots inside actions are inputs and outputs. Parameters are not shown.

integers. Further, we assume there are only 3 actions the planner may take: compose two regions horizontally (*comp2h*) or vertically (*comp2v*), and compose 4 regions (*comp4*) as in Figure 1.

A problem instance we consider here consists of some unit squares; that is, squares of $\langle ul, lr\rangle$ where $ul.x + 1 = lr.x$ and $ul.y + 1 = lr.y$. The goal is to compose a region covering $\langle(0, 0), (3, 2)\rangle$. As in Figure 2, the region $\langle(0, 0), (3, 2)\rangle$ consists of 6 unit squares denoted by $B_1, B_2, ..., B_6$. For example, B_1 refers to the region $\langle(0, 0), (1, 1)\rangle$, and B_1B_2 together refer to the region $\langle(0, 0), (2, 1)\rangle$. These unit squares are available in the initial state.

The planning graph construction starts with the planner goal. For the goal, the planner finds 3 possible actions that can achieve the goal and add 3 nodes in the graph linked to the goal. By regressing the goal through each supporting action, the planner identifies the subgoals to be achieved. This process continues until all subgoals added to the graph have support either from supporting actions or from the initial state. The planning graph created by the planner is in Figure 2.

At a high level, the planner finds a lifted plan by selecting subgoals and actions, shown in Figure 2 as a path from the initial state to the goal with dark arrows. This plan may not be executable because actions are not grounded. For example, the action *comp4* is selected because it has support from the initial state and it supports the action *comp2h*, but its parameters are not determined yet. The constraint solver then finds values for action parameters, which are shown in Figure 2 as groups of shaded rectangles.

It turns out that finding a lifted plan is a relatively easy task because it is a problem of finding a consistent assignment to a small number of variables in a very big constraint problem; whereas finding values for action parameters is a difficult CSP search problem. To address the issue, we developed a graph-based search algorithm, which is discussed in the rest of the paper.

3 Graph-based Backtracking

A constraint satisfaction problem (CSP) consists of variables, domains that contain possible values the variables may take, and constraints that limit the values the variable can take simultaneously. In finite domain CSPs, a constraint over a variable subset can be represented extensionally as a subset of the Cartesian product of the domains of variables in the constraint. However, in the data production domain we are interested in, the CSPs derived from the planning problem contain variables that usually have infinite domains. An infinite domain is represented as an an interval (for numeric types), regular expression (for string types), or symbolic set (for object types). Because of infinite domains, the constraints are not represented extensionally as relations, but as procedures [Jónsson, 1996]. A procedural constraint consists of a set of variables (the scope) and a procedure (i.e., an `enforce()` method) that enforces the constraint by eliminating inconsistent values from the domains of variables in the scope. If enforcing a constraint results in an empty variable domain, `enforce()` returns failure, indicating the violation of the constraint.

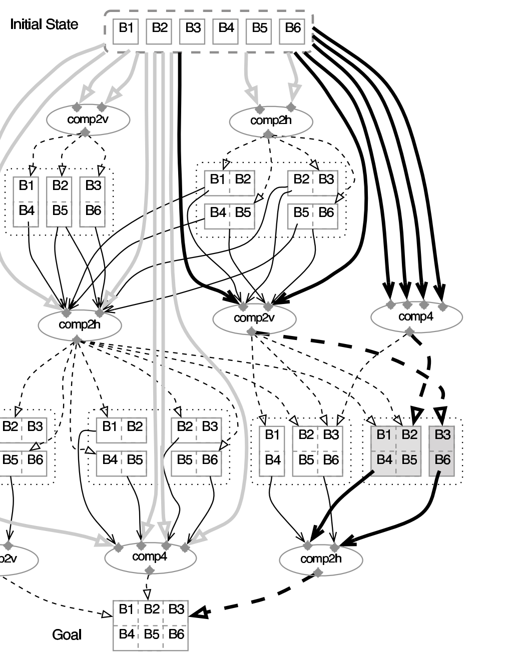


Figure 2: A planning graph: objects in a dotted rectangles are inputs to an action; an object divided by dashed lines is a composed object; single objects are available in the initial state.

3.1 Structure-based Decomposition

Solving a CSP, in general, is NP-complete. However, many practical problems possess certain properties that allow tractable solutions. A class of structure-based CSP-solving algorithms, called decomposition algorithms, has been developed [Gottlob, 2000; Gyssens, Jeavons, & Cohen, 1994; Dechter, 1990; Dechter & Pearl, 1989]. Decomposition algorithms attempt to find solutions by decomposing a CSP into several simply connected sub-CSPs based on the underlying constraint graph and then solving them separately. Once a CSP is decomposed into a set of sub-CSPs, all solutions for each sub-CSP are found. Then a new CSP is formed where the original variable set in each sub-CSP is taken as a singleton variable. The technique aims at decomposing a CSP into sub-CSPs such that the number of variables in the largest sub-CSP is minimal and the newly formed CSP has a tree-structured constraint graph. In this way, the time and space complexity of finding all solutions for each sub-CSP is bounded, and the newly formed CSP has backtrack-free solutions. The complexity of a decomposition algorithm is exponential in the size of the largest sub-CSP. The class of CSPs that can be decomposed into sub-CSPs such that their sizes are bounded by a fixed number k is tractable and can be solved by decomposition in polynomial time. This is the strength of CSP decomposition. A fatal weakness of CSP decomposition, however, is that the decomposition is not applicable to solving a CSP that is not decomposable, that is, its decomposition is itself. A secondary drawback of CSP decomposition is that, even if the CSP is decomposable, finding all solutions for all the sub-CSPs is unnecessary and inefficient.

We propose a graph based backtracking (GBT) algorithm, based on our previous work in [Pang & Goodwin, 2003], to address these two issues: decomposability and efficiency of solving sub-problems.

3.2 Algorithms

The original GBT [Pang & Goodwin, 2003] uses the same strategy used by decomposition algorithms to decompose the constraint graph into a tree of subgraphs and then performs backtracking at two nested levels: *inner-backtracking* inside a subgraph and *outer-backtracking* following the subgraph tree obtained from the graph decomposition. It shares the merits of CSP decomposition and it does not need to find all solutions for all the sub-CSPs. However, its performance and applicability largely depend on the underlying graph decomposition. In the planning problem at hand, we tried a variety of graph decomposition algorithms and were unable to decompose the constraint graph into a tree of subgraphs in a satisfactory way, even for simplified problems. It is still an on-going research effort to evaluate the process of translating the planning problem to a CSP aiming at optimizing the constraint problem in terms of its size and structural properties.

As an alternative, we decompose the CSP into sub-CSPs based on the planning graph instead of the constraint graph, each sub-CSP containing a group of variables that are relevant to a node in the planning graph representing a lifted action. In most of the cases, the sub-CSPs may not form a tree, which makes the traditional decomposition methods inapplicable. However, the GBT algorithm can be adapted easily: outer-backtracking is performed to select the planner subgoals and actions following the planning graph, inner-backtracking to find values for action parameters by solving the associated sub-CSP.

The planning-graph-based backtracking algorithm is outlined in Algorithms 1 and 2. At the high level, the planner performs Best-First Search (BFS) to select the planner subgoals and actions achieving the subgoals. Once an action is chosen for a subgoal, it collects a subset of variables relevant to the action and calls a constraint solver, SBT, to find a consistent assignment for the collected variables. SBT performs a local backtracking to search for a solution to the sub-problem that is also consistent with solutions to the sub-problems preceding the current one. If SBT fails, the high-level search BFS takes control, tries another action for the current selected subgoal or backtracks to a previously selected subgoal. At the end of the selection of subgoals and actions, SBT is called again to find values for certain variables that may have been missed during the previous search.

Both algorithms interleave search with propagation, which is a process of continuously enforcing constraints as long as variable domains change. The propagation performs a partial *generalized arc-consistency* (GAC)[1][Bessiere & Ch, 1997; Katsirelos & Bacchus, 2001], and it is an essential part of solving the constraint problem, not only because it reduces the search space by eliminating some inconsistent values, but also because the constraint problem at hand contains variables with infinite domains which cannot be enumerated by search. If enforcing a constraint fails, `propagate()` returns `failure`, which implies that the current assignment of values to variables is inconsistent. Because the propagation is not limited to a sub-problem even if it is invoked by the SBT solving the sub-problem, it ensures the solutions to local sub-problems are globally consistent with respect to the variables in the other solved sub-problems.

3.3 The Example Again

We take a look at the simplified example again and describe how the graph based search algorithm works.

The task is to make a region covering $\langle(0,0),(3,2)\rangle$, which consists of regions referred to as$B_1, B_2, ..., B_6$, all available from the initial state. At the beginning of the planning search, the active goals G' contains only the planner goal of making $\langle(0,0),(3,2)\rangle$. Ignoring the planning heuristics, we assume that the planner chooses the action *comp2h*, which composes two regions horizontally. These two regions are the parameters of the action that are not determined in BFS. The inner-backtracking solver SBT is created with these parameters and it is called to find values for the parameters. It quickly finds the values for the parameters; that is, two regions $\langle(0,0),(2,2)\rangle$ and $\langle(2,0),(3,2)\rangle$ for output parameters of action *comp2h*, and it also remembers its current status so that when it backtracks, it can find the next solution (in this case it is the regions $\langle(0,0),(1,2)\rangle$ and $\langle(1,0),(3,2)\rangle$, (see Figure 2) . The planner adds the two subgoals, making the region $\langle(0,0),(2,2)\rangle$ and making

[1]We call it partial GAC for two reasons: 1) not every constraint procedure enforces the GAC; and 2) not every constraint is enforced in the propagation.

Algorithm 1 BFS

Given a set of subgoals in the lifted planning graph G and a family of action sets $A = \{A(g)|g \in G\}$, each $A(g)$ is a set of actions achieving subgoal g. Let $G' \subseteq G$ be a set of active subgoals to be achieved (initially, the goals in the goal state), $P = (X, D, C)$ the CSP derived from the lifted planning graph, and X' a subset of searchable variables:

```
BFS($G, A, P, G'$)
  1. while ($G' \neq \emptyset$) do
     (a) $g \leftarrow$ a goal removed from $G'$
     (b) for each action $a \in A(g)$
         i. if (propagate($P, \{a\}$) returns failure)
            continue for the next action
        ii. $X' \leftarrow$ variables relevant to $a$
       iii. while (SBT($P, X'$) returns success) do
            A. $G_a \leftarrow$ conditions of $a$
            B. add $G_a$ to $G'$ and sort $G'$
            C. if (GBFS($G, A, P, G'$) returns success)
               return SBT($P, X$)
        iv. continue for the next action
     (c) return failure
  2. return success
```

Algorithm 2 SBT

Given a CSP $P = (X, D, C)$, and let $X' \subseteq X$ be a set of searchable variables:

```
SBT($P, X'$)
  1. if ($X' = \emptyset$) return success
  2. select $x_i \in X$
  3. for each value $v \in d(x_i)$
     (a) $x_i \leftarrow v$
     (b) if (propagate($P$,{$x_i$}) returns success)
         i. update $X'$
        ii. if (SBT($P$, $X'$) returns success)
            return success
  4. return failure
```

the region $\langle(2, 0), (3, 2)\rangle$, to the active goals, and then continues recursively with the next best goal, which is one of the newly added two subgoals.

4 TOPS Application

We have applied the DoPPLER planner to the Terrestrial Observation and Prediction System (TOPS, http://ecocast.arc.nasa.gov) [Nemani *et al.*, 2002], an ecological forecasting system that assimilates data from Earth-orbiting satellites and ground weather stations to model and forecast conditions on the surface, such as soil moisture, vegetation growth and plant stress. The planner identifies the appropriate input files and sequences of operations needed to satisfy a data request, executes those operations on a remote TOPS server, and displays the results, quickly and reliably.

We have developed a TOPS planning domain, which specifies the data operations and data object types in TOPS. Data operations include running simulation-based models, reprojection, scaling, and construction of color composites, mosaics, and animations, etc. To create a planning problem instance (i.e., a TOPS task), the user needs only to specify the planner goal, which is a description of a desired data product. A sample TOPS task would be something like "display Gross Primary Production (GPP) for continental US on May 5th, 2004".

The motivation of developing the graph based planning search is to speed up the search process so that the planner can produce the requested data product within a time limit acceptable to the user. Even though it is difficult for us to compare DoPPLER planner with publicly available planners, which cannot handle data-processing problems, we have compared DoPPLER to itself by turning on or off the inner-backtracking SBT. Without inner-backtracking SBT, for most of the TOPS tasks, it usually takes a few tries with different variable ordering heuristics to solve a problem; sometimes it fails within a specified time limit (e.g., 5 minutes). With inner-backtracking SBT turned on, the same TOPS tasks can be solved quickly without trying the additional variable ordering heuristics. However, we are currently conducting experiments on more TOPS tasks and artificial problems like the one in Section 2.5.

5 Conclusions

We have discussed the data production problem and how we reduce it to planning and solve the planning problem with a constraint search and propagation approach. A key element of our approach is the lifted planning graph, which we use as a basis for our CSP encoding, and use further to guide the planning and constraint search. The graph-based backtracking algorithm presented here has proved to be effective in our planner; it is also a general CSP solver that we intend to evaluate further on structured or semi-structured problems and to compare to other search and decomposition methods.

There has been little work in planner-based automation of data production. Two notable exceptions are Collage [Lansky, 1998] and MVP [Chien *et al.*, 1997]. Both of these planners were designed to provide assistance with data analysis tasks in which a human was in the loop, directing the planner. In contrast, our planner does not require human interaction, since domains like TOPS require data production to be entirely automated; there is simply too much data for human interaction to be practical. Pegasus [Blythe *et al.*, 2003] is a workflow planning system for computation grids, a problem similar to ours, though their focus is on mapping pre-specified workflows onto a specific grid environment, whereas our focus is on generating the workflows.

References

[Bessiere & Ch, 1997] Bessiere, C., and Ch, J. 1997. Arc-consistency for general constraint networks: Preliminary results. In *Proceedings of IJCAI-97*, 398–404.

[Blum & Furst, 1997] Blum, A., and Furst, M. 1997. Fast planning through planning graph analysis. *AIJ* 90(1–2):281–300.

[Blythe *et al.*, 2003] Blythe, J.; Deelman, E.; Gil, Y.; Kesselman, C.; Agarwal, A.; Mehta, G.; and Vahi, K. 2003. The role of planning in grid computing. In *Proc. 13th Intl. Conf. on Automated Planning and Scheduling (ICAPS)*.

[Chien *et al.*, 1997] Chien, S.; Fisher, F.; Lo, E.; Mortensen, H.; and Greeley, R. 1997. Using artificial intelligence planning to automate science data analysis for large image database. In *Proc. 1997 Conference on Knowledge Discovery and Data Mining*.

[Dechter & Pearl, 1989] Dechter, R., and Pearl, J. 1989. Tree clustering for constraint networks. *Artificial Intelligence* 38:353–366.

[Dechter, 1990] Dechter, R. 1990. Enhancement schemes for constraint processing: backjumping, learning, and cutset decomposition. *Artificial Intelligence* 41:273–312.

[Do & Kambhampati, 2001] Do, M., and Kambhampati, S. 2001. Planning as constraint satisfaction: Solving the planning graph by compiling it into CSP. *Artificial Intelligence* 132:151–182.

[Golden & Frank, 2002] Golden, K., and Frank, J. 2002. Universal quantification in a constraint-based planner. In *AIPS02*.

[Golden, 2002] Golden, K. 2002. DPADL: An action language for data processing domains. In *Proceedings of the 3rd NASA Intl. Planning and Scheduling workshop*, 28–33. to appear.

[Golden, 2003] Golden, K. 2003. An domain description language data processing. In *ICAPS 2003 Workshop on the Future of PDDL*.

[Gottlob, 2000] Gottlob, G. 2000. A comparison of structural CSP decomposition methods. *Artificial Intelligence* 124:243–282.

[Gyssens, Jeavons, & Cohen, 1994] Gyssens, M.; Jeavons, P.; and Cohen, D. 1994. Decomposing constraint satisfaction problems using database techniques. *Artificial Intelligence* 66:57–89.

[Jónsson, 1996] Jónsson, A. 1996. *Procedural Reasoning in Constraint Satisfaction*. Ph.D. Dissertation, Stanford University.

[Katsirelos & Bacchus, 2001] Katsirelos, G., and Bacchus, F. 2001. GAC on conjunctions of constraints. In *CP-2001*.

[Lansky, 1998] Lansky, A. 1998. Localized planning with action-based constraints. *Artificial Intelligence* 98(1–2):49–136.

[Lopez & Bacchus, 2003] Lopez, A., and Bacchus, F. 2003. Generalizing graphplan by formulating planning as a CSP. In *Proceedings of IJCAI-2003*.

[Nemani *et al.*, 2002] Nemani, R.; Votava, P.; Roads, J.; White, M.; Thornton, P.; and Coughlan, J. 2002. Terrestrial observation and predition system: Integration of satellite and surface weather observations with ecosystem models. In *Proceedings of the 2002 International Geoscience and Remote Sensing Symposium (IGARSS)*.

[Pang & Goodwin, 2003] Pang, W., and Goodwin, S. D. 2003. A graph based backtracking algorithm for general CSPs. In *Proceedings of 6th Canadian Conference on Artificial Intelligence (CAI-2003)*, 114–128.

[Penberthy & Weld, 1992] Penberthy, J., and Weld, D. 1992. UCPOP: A sound, complete, partial order planner for ADL. In *Proc. 3rd Int. Conf. Principles of Knowledge Representation and Reasoning*, 103–114.

[Smith, Frank, & Jónsson, 2000] Smith, D.; Frank, J.; and Jónsson, A. 2000. Bridging the gap between planning and scheduling. *Knowledge Engineering Review* 15(1):61–94.

[van Beek & Chen, 1999] van Beek, P., and Chen, X. 1999. CPlan: A constraint programming approach to planning. In *Proceedings of AAAI-99*.

Combining constraint processing and pattern matching to describe and locate structured motifs in genomic sequences

Patricia Thébault, Simon de Givry, Thomas Schiex, and Christine Gaspin
{pat,degivry,tschiex,gaspin}@toulouse.inra.fr
INRA Biometrics and Artificial Intelligence
Toulouse, France

Abstract

In molecular biology and bioinformatics, searching RNA gene occurrences in genomic sequences is a task whose importance has been renewed by the recent discovery of numerous functional RNA, often interacting with other ligands. Even if several programs exist for RNA motif search, no program exists that can represent and solve the problem of searching for occurrences of RNA motifs **in interaction with other molecules**.

In this paper, we present a CSP formulation of this problem. We represent such RNA as structured motifs that occur on more than one sequence and which are related together by possible hybridization. Together with pattern matching algorithms, constraint satisfaction techniques have been implemented in a prototype MilPat and applied to search for tRNA and snoRNA genes on genomic sequences. Results show that these combined techniques allow to efficiently search for interacting motifs in large genomic sequences and offer a simple and extensible framework to solve such problems.

1 Introduction

Our understanding of the role of RNA has changed in recent years. Firstly considered as being simply the messenger that converts genetic information from DNA into proteins, RNA is now seen as a key regulatory factor in many of the cell's crucial functions, affecting a large variety of processes including plasmid replication, phage development, bacterial virulence, chromosome structure, DNA transcription, RNA processing and modification, development control and others (for review [Storz, 2002]). Consequently, the systematic search of non-coding RNA (ncRNA) genes, which produce functional RNAs instead of proteins, represents an important challenge.

RNA sequences can be considered as texts over the four letter alphabet $\{A,C,G,U\}$. Unlike double-stranded DNA, RNA molecules are almost exclusively found in an oriented (left or 5' to right or 3') single-stranded form and often fold into more complex structures than DNA by making use of so called complementary internal sequences. This characteristic allows different regions of the same RNA strand (or of several RNA strands) to fold together via a variety of interactions to build structures that are essential for the biological function. The level of organization relevant for biological function corresponds to the spatial organization of the entire nucleotides chain and is called the tertiary structure. The most prevalent interactions which stabilize folded molecules are stacking and hydrogen bonding between nucleotides on strands oriented in antiparallel directions. Similarly to what exists in DNA, hydrogen bonds appear mostly between specific pairs of nucleic acids to form *G–C* and *C–G* or *A–U* and *U–A* bonds. Therefore the interactions inside an RNA molecule usually involve one part of a molecule and the nucleic acid complement of a another part of the same molecule (for example, 5'-ACUCGA-3' and 5'-UCGAGU-3'), and the two antiparallel regions bind together.

All together, these interactions define the molecule three-dimensional structure which is essential to characterize its function and interactions with other molecules. Due to the difficulty of determining such three dimensional RNA structures, one first explores the so-called RNA secondary structure, a simplified model of the RNA three dimensional tertiary structure.

This secondary structure gives only a subset of those interactions represented by *C–G*, *G–C*, *A–U*, and *U–A* pairs and provides an important constraint for determining the three dimensional structure of RNA molecules.

An RNA molecule secondary structure can be represented on a circular planar graph where the *N* nucleotides of the sequence are represented as vertices and are connected by edges representing either (along the circle) covalent bonds between successive nucleotides in the RNA sequence or (inside the circle) hydrogen bonds between nucleotides from different regions. Such a graph gives rise to characteristic secondary structural elements (see Fig. 1) such as helices (a succession of paired nucleotides), and various kinds of loops (unpaired nucleotides surrounded by helices).

A more complete definition of secondary structure of RNA allows for crossing edges in the representation graph making possible the representation of another type of helix usually called a pseudoknot (see Fig. 1). RNA structures can also include nucleotide triples inside triple helices...

This definition extends the usual definition which is often limited to planar structures (therefore excluding pseudo-

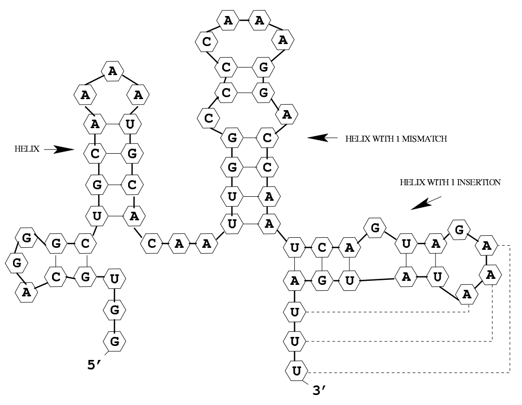


Figure 1: A representation of a secondary structure as a planar graph. Thick edges represent covalent bonds. Thin edges represent hydrogen interactions. Dotted edges represent a pseudoknot. Helices may contain local mismatches which cover three different types of errors which are: insertions of nucleotide(s), also called bulges when only on one side, deletions when the insertion is on the opposite side of the helix and internal loops, when nucleotides are located on both sides of the same helix. Insertion and deletion of nucleotides are considered as symmetric operations, an insertion on one side corresponding to a deletion on the other complementary region.

knots and multiple helices) and is always restricted to intra-sequences interactions.

In this paper, we use the extended definition where the secondary structure of an RNA gene is defined as the set of paired nucleotides which appear in the folded RNA, including possible pseudoknots, triple helices but also duplexes which are possible bindings forming helices with other RNA molecules.

Screening a sequence database with tools designed for sequence similarity search quickly reveals similarities between the query sequence and a range of database sequences. This can be achieved for ribosomal rRNA sequences and other ncRNAs recently reported in the literature (although it is difficult to establish the beginning and end of the RNA in question). But the nucleotide sequence of the RNA itself is poorly conserved, the observation that the functionally important structural regions are usually conserved in an RNA family (see for example Fig. 2) allows one to search for those elements that characterize the family more precisely.

Thus, the information contained both in the sequence itself and the secondary (tertiary) structure can be viewed as a biological signal to exploit and search for. Thus, whatever the method, it appears necessary to include both conserved primary sequence elements and higher order structure elements as signals to screen for. These common structural characteristics can be captured by a signature that represents the structural elements which are conserved inside a set of related RNA molecules.

We focus here on the problem of searching for new members of a gene family given their common signature. Solving this problem requires (1) to be able to formalize what a signature is and what it means for such a signature to occur in a sequence (2) to design algorithms and data-structures that can efficiently look for such occurrences in large sequences. For sufficiently general signatures, this is an NP-complete problem [Vialette, 2004] that combines combinatorial optimization and pattern matching issues.

Traditionally, two types of approaches have been used for RNA gene finding: signatures can be modelled as stochastic context free grammars (excluding pseudo-knots or complex structures) and then searched using relatively time consuming dynamic programming based parsers. This is e.g. used in [Sakakibara *et al.*, 1994; Eddy and Durbin, 1994] for RNA genes or in [Bockhorst and Craven, 2001] for terminators.

Another approach defines a signature as a set of interrelated motifs. Occurrences of the signature are sought using simple pattern-matching techniques and exhaustive tree search. Such programs include RnaMot [Gautheret *et al.*, 1990], RnaBob [Eddy, 1996], PatScan [Dsouza *et al.*, 1997], Palingol [Billoud *et al.*, 1996] and RnaMotif [Macke *et al.*, 2001]. Although most allow pseudo-knots to be represented, they have very variable efficiencies and are all restricted to single RNA molecule signatures.

In this paper, we clearly separate the combinatorial aspects from the pattern matching aspect by modelling a signature as a CSP. The CSP model captures the combinatorial features of the problem while the constraints use pattern matching techniques to enhance efficiency. This combination offers an elegant and simple way to describe several RNA motifs in interaction and a general purpose efficient algorithm to search for occurrences of such motifs.

2 Methods

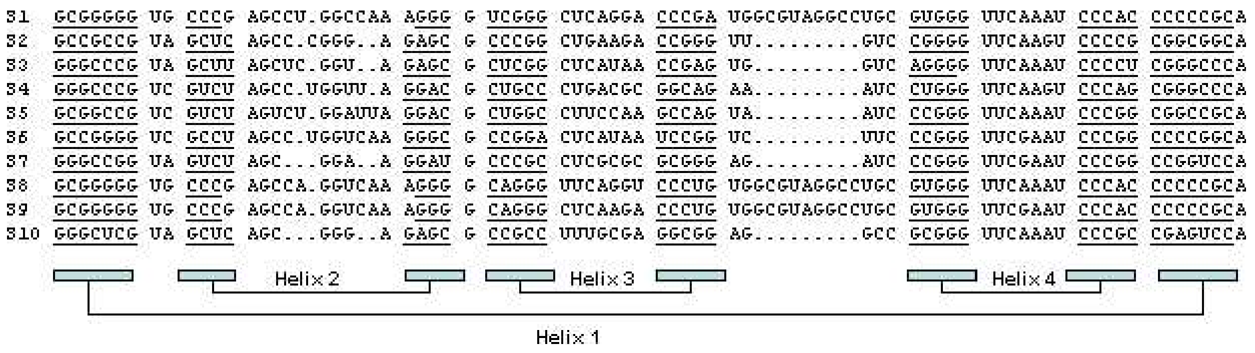
The CSP formalism (see e.g. [Dechter, 2003]) is a powerful and extensively used framework for describing combinatorial search problems in artificial intelligence and operations research. This is usually well adapted to the definition of mathematical problems raised by molecular biology (see [Gaspin and Westhof, 1994; Muller *et al.*, 1993; Altman *et al.*, 1994; Major *et al.*, 1991; Barahona and Krippahl, 1999]) and has been used to model the structured motif search problem in [Eidhammer *et al.*, 2001; Policriti *et al.*, 2004].

2.1 Structured motifs as CSPs

The elements that may characterize an RNA gene family are usually described:

- in terms of the gene sequence itself (e.g. it must contain some possibly degenerated pattern);
- in terms of the structures the sequence creates: loops, helices, hairpins and possible duplexes with other molecules;
- by specifying how these various elements are positioned relatively to each other.

A possible occurrence of such a structured motif on a genomic sequence can be described by the positions of the various elements on the genomic sequence. A true occurrence is



```
S1  GCGGGGG UG CCCG AGCCU.GGCCAA AGGG G UCGGG CUCAGGA CCCGA UGGCGUAGGCCUGC GUGGG UUCAAAU CCCAC CCCCCGCA
S2  GCCGCCG UA GCUC AGCC.CGGG..A GAGC G CCCGG CUGAAGA CCGGG UU.........GUC CGGGG UUCAAGU CCCCG CGGCGGCA
S3  GGGCCCG UA GCUU AGCUC.GGU..A GAGC G CUCGG CUCAUAA CCGAG UG.........GUC AGGGG UUCAAAU CCCCU CGGGCCCA
S4  GGGCCCG UC GUCU AGCC.UGGUU.A GGAC G CUGCC CUGACGC GGCAG AA.........AUC CUGGG UUCAAGU CCCAG CGGGCCCA
S5  GCGGCCG UC GUCU AGUCU.GGAUUA GGAC G CUGGC CUUCCAA GCCAG UA.........AUC CCGGG UUCAAAU CCCGG CGGCCGCA
S6  GCCGGGG UC GCCU AGCC.UGGUCAA GGGC G CCGGA CUCAUAA UCCGG UC.........UUC CCGGG UUCGAAU CCCGG CCCCGGCA
S7  GGGCCGG UA GUCU AGC...GGA..A GGAU G CCCGC CUCGCGC GCGGG AG.........AUC CCGGG UUCGAAU CCCGG CCGGUCCA
S8  GCGGGGG UG CCCG AGCCA.GGUCAA AGGG G CAGGG UUCAGGU CCCUG UGGCGUAGGCCUGC GUGGG UUCAAAU CCCAC CCCCCGCA
S9  GCGGGGG UG CCCG AGCCA.GGUCAA AGGG G CAGGG CUCAAGA CCCUG UGGCGUAGGCCUGC GUGGG UUCGAAU CCCAC CCCCCGCA
S10 GGGCUCG UA GCUC AGC...GGG..A GAGC G CCGCC UUUGCGA GGCGG AG.........GCC GCGGG UUCAAAU CCCGC CGAGUCCA
```

Figure 2: Alignment of a subset of ten sequences of the tRNA family extracted from the RFAM RNA database (http://www.sanger.ac.uk/cgi-bin/Rfam). Each line gives the tRNA gene sequence. Both sides of each helix are underlined for each sequence of the alignment. Consensus helices are identified by boxes at the end of the alignment. tRNA genes include four helices corresponding respectively to helix 1 called A-stem (7 nucleotide pairs), helix 2 called D-stem (from 3 to 4 nucleotide pairs), helix 3 called C-stem (5 nucleotide pairs) and the last fourth helix, called, T-stem (5 nucleotide pairs). Six loops corresponding respectively to the single strand between A-stem and D-stem (sequence UN with U invariant), D-loop (4 to 14 nucleotides), the single strand between D-stem and C-stem (one nucleotide), C-loop (6 to 60 nucleotides), the single strand between C-stem and T-stem (also called V-loop, 2 to 22 nucleotides), T-loop (NUC) allow to build a signature of the family. Note several hundred tRNA sequences are now available from biological databanks (see in particular RFAM).

such that the required patterns, structures actually appear in the genomic sequence and are correctly positioned relatively to each other. Note that a genomic sequence is represented as a string defined over the RNA alphabet $\{A, U, G, C\}$.

A natural CSP model emerges from this description: the variables will represent the positions on the nucleotide sequence of the elements of the description. More formally, each variable $x_i \in X$ will represent a position on an associated RNA sequence (denoted t_i). The initial domain of variable x_i, unless otherwise stated, will therefore be equal to $[1, |t_i|]$. In order to represent information on required patterns, structures and on relative positions of these elements, constraints will be used. To describe a constraint we separate the *variables* $x_i, ..., x_j$ involved in the constraint (its scope) and possible extra *parameters* $p_1, ..., p_k$ that influence the actual combination of values that are authorized by the constraint. Such a constraint will be denoted as $\mathtt{name}[p_1, ..., p_k](x_i, ..., x_j)$. We now introduce the basic constraint types which are useful for RNA signature expression:

composition$[\mathsf{word}, \mathsf{error}, \mathsf{type}_{\mathsf{err}}](x_i)$
this unary constraint is satisfied iff some given sequence (a pattern) occurs at position x_i on sequence t_i. The pattern that must occur is specified by the following constraint parameters:

- word is a word on the so-called IUPAC alphabet which includes meta-characters that match several characters of the RNA alphabet.
- error specifies the maximum number of tolerated mismatches between an occurrence and the specified string.
- type$_{\mathsf{err}}$ indicates if the error count is interpreted under the Hamming or Levenstein distance metric [Smith and Waterman, 1981].

An example of possible use of this constraint is illustrated in Fig. 3(1) where variable x_1 is constrained to a position where the AGGGCUAG pattern must appear with no error. A position satisfying this constraint (or occurrence of the pattern) is indicated by the arrow.

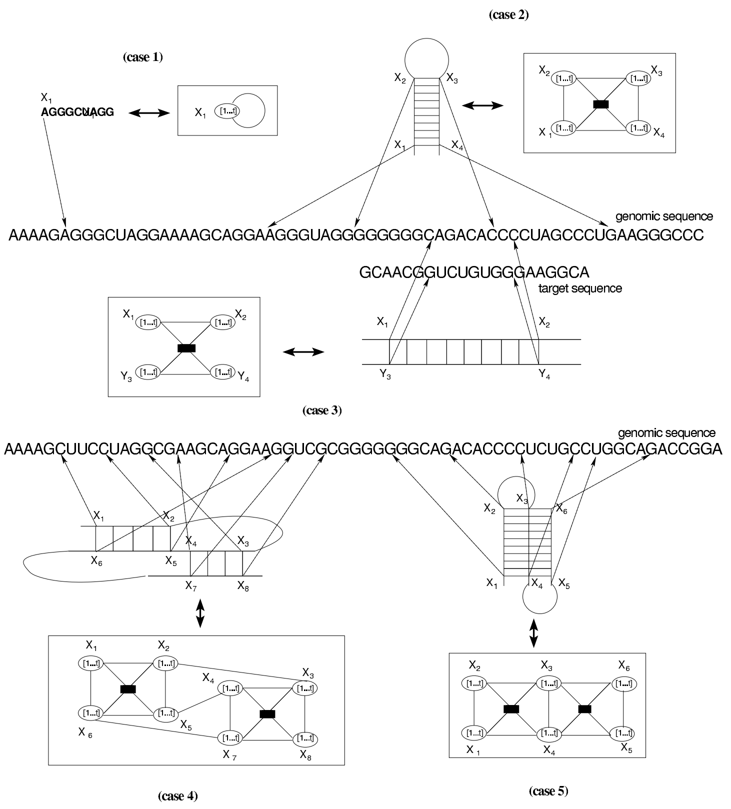
distance$[\mathsf{l}_{\mathsf{min}}, \mathsf{l}_{\mathsf{max}}](x_{i_1}, x_{i_2})$
this binary constraint is use to enforce the relative position of elements. It is satisfied iff

$$\mathsf{l}_{\mathsf{min}} \leq x_{i_2} - x_{i_1} \leq \mathsf{l}_{\mathsf{max}}$$

The parameters $\mathsf{l}_{\mathsf{min}}, \mathsf{l}_{\mathsf{max}}$ specify the bounds for the difference between the two position variables. It is a simple usual arithmetic constraint.

helix$[\mathsf{rule}, \mathsf{error}, \mathsf{type}_{\mathsf{err}}, \mathsf{l}_{\mathsf{min}}, \mathsf{l}_{\mathsf{max}}, \mathsf{b}_{\mathsf{min}}, \mathsf{b}_{\mathsf{max}}](x_{i_1}, x_{i_2}, x_{i_3}, x_{i_4})$
this 4-ary constraint is used to enforce the existence of an helix between the sequence regions delimited by $[x_{i_1}, x_{i_2}]$ and $[x_{i_3}, x_{i_4}]$. This constraint assumes that the four variables are related to the same sequence (it models intra-sequence interactions) and each region represents a substring of this sequence. The length and distances between these regions are also constrained. The constraint must be specified by the following parameters:

- rule: a binary relation on the RNA alphabet that characterizes which pairs of nucleotides are allowed inside an helix. For an RNA helix, one typically uses Watson-Crick (A−U and G−C) pairs,possibly extended with Wobble (G−U) pairing.
- error: the maximum number of tolerated mismatches between the two regions (nucleotides that do not satisfy the previous paring relation).
- type$_{\mathsf{err}}$: the Hamming or Levenstein distance metric for error counts.



(case 1)
(case 2)
(case 3)
(case 4)
(case 5)
genomic sequence
target sequence
AAAAGAGGGCUAGGAAAAGCAGGAAGGGUAGGGGGGGGCAGACACCCCUAGCCCUGAAGGGCCC
GCAACGGUCUGUGGGAAGGCA
AAAAGCUUCCUAGGCGAAGCAGGAAGGUCGCGGGGGGGCAGACACCCCUCUGCCUGGCAGACCGGA

Figure 3: Basic constraints. (case 1): occurrence of a pattern at one position (variable). The constraint graph contains one variable with a unary constraint represented by a loop. (case 2): an helix and a loop defined by two related segments separated by specified lengths. The constraint graph contains four variables, four implicit distance constraints represented by edges and one hyper-edge (for the helix constraint) connecting all four variables with a rectangle in the middle. (case 3): a duplex composed of two independent substrings (from two sequences). The constraint graph is similar to the previous one (two distance constraints are removed). (cases 4 and 5): two helix constraints can describe a pseudo-knot (4) or a triple helix (5).

- $\mathsf{l}_{\min}, \mathsf{l}_{\max}$: the interval specifying possible lengths of the two substrings.

- $\mathsf{b}_{\min}, \mathsf{b}_{\max}$: the interval specifying the possible distance between the two substrings (*i.e.*, $x_{i_3} - x_{i_3}$).

This constraint is illustrated in Fig. 3(2), involving variables x_1, x_2, x_3 and x_4. Assuming Watson-Crick pairing, no error and suitable lengths, the constraint is satisfied for the values indicated by arrows on the sequence below.

`duplex`$[\mathsf{l}_{\min}, \mathsf{l}_{\max}](x_1, x_2, y_3, y_4)$

this 4-ary constraint is used to enforce the existence of a (Watson-Crick based) duplex between the regions delimited by $[x_1, x_2]$ and $[y_3, y_4]$. Although semantically equivalent to the previous one, it does not assume that the two substrings represented by the two regions belong to the same sequence. This has important computational impact. Only Watson-Crick pairing is considered. This constraint is used to model RNA-RNA interactions between possibly different molecules.

- $\mathsf{l}_{\min}, \mathsf{l}_{\max}$: the interval specifying possible lengths of the two substrings.

The constraint is illustrated in Fig. 3(3) involving x_1, x_2 (on one sequence) and y_3 and y_4 on another sequence. Values satisfying the constraint (an occurrence) is indicated by the arrows.

Note that together these constraints can describe more complex structures like pseudo-knots (Fig. 3(4)), triple helices (Fig. 3(5)), and so on.

The flexibility of the CSP formalism using simply the four previous basic constraints can be illustrated on famous RNA gene families. The **tRNA** signature is represented in Fig. 4 where tRNA genes include four helices. The corresponding CSP is build from 16 variables (the variable numbering follows the 3' → 5' orientation) with 15 `distance` constraints (one constraint between each successive pair of variables), 2 `composition` constraints and 4 `helix` constraints.

The same process can be applied to the **snoRNA** signature depicted in Fig. 5. snoRNA genes include a C box (`RUGAUGA`) with one error allowed, a single strand from 22 to 44 nucleotides, a duplex with a target RNA from 9

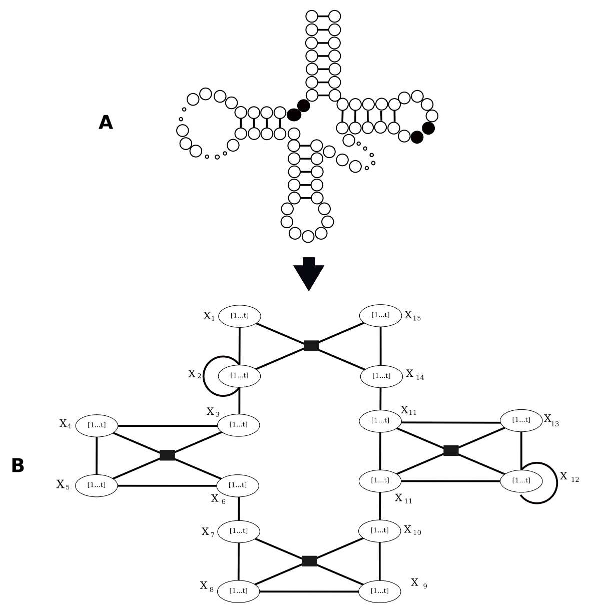


Figure 4: (A) Signature of tRNA genes family. White circles : nucleotides with unknown composition, black circles : known composition, little circles : number of nucleotides given by an interval, and edge : interaction between two nucleotides. (B) Corresponding CSP model.

to 15 nucleotides and a D box (CUGA) with one error allowed. The corresponding CSP is build from 4 variables corresponding to positions on the genomic sequence and a pair of additional variables associated with the target RNA. The first set of variables is linked with 3 `distance` constraints and 2 `composition` constraints. The second set with one `distance` constraint. Both sets are connected through one `duplex` constraint.

2.2 Algorithms and implementation

Given such CSPs, our problem is to find **all solutions**. Compared to usual applications of the CSP formalism, this one is characterized by the potential huge domain size (the length of a complete pseudo-molecule can be greater than several million of nucleotides) and its specific constraint types (except for the `distance` constraint which is a usual arithmetic constraint). For efficiency and memory space reasons, it is not possible to represent variable domains exhaustively and to enforce arc consistency on them. As it is done in Constraint Programming [Dechter, 2003], we represent the domain of each variable x_i by an interval $[lb_i, ub_i]$ and reason only on domain bounds as done in arc-bound consistency [Lhomme, 1993]. This limited bound filtering is done at each node inside a usual tree search algorithm. For n-ary constraints, the typical form of local consistency used enforces the fact that the bounds in the domain of one variable in the constraint scope must participate in at least one tuple that is authorized by the constraint and the other domains. The exploration method we used is a depth-first search algorithm with a refutation mechanism (during backtracks, it propagates the removal of values

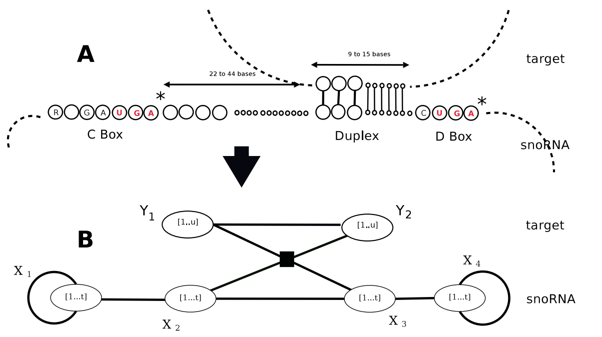


Figure 5: Signature of snoRNA genes family including its target interaction.

already explored).

Dedicated constraint propagation

For each type of constraint, we developed specific filtering algorithm using appropriate pattern matching algorithms (except for the `distance` constraint where we used the filtering algorithm described in [Hentenryck *et al.*, 1992]):

- `composition`[...](x_i): to enforce arc consistency on the lower bound of the domain of x_i, one can simply update this to the position of the first occurrence of the pattern after lb_i in the text t_i. To find this occurrence, the algorithm of Baeza-Yates and Manber [Baeza-Yaltes and Gonnet, 1992; Wu and Manber, 1991] is used. This algorithm is based on a boolean representation of the search state and exploit the intrinsic parallelism of bitwise logical operations in modern CPU. It has a linear complexity for exact string search and a complexity in $O(m \times n)$ for the Levenstein distance (m being the length of the text and n that of the pattern sought). A similar processing can be done on the other bound (but is not used in our prototype).
- `helix`[...]($x_{i_1}, x_{i_2}, x_{i_3}, x_{i_4}$): Consider for example variable x_{i_1}. To filter x_i domain, we must find the first helix (a support) that satisfies the parameters of the constraint. By first we mean the helix with the smallest position of the 5' extremity of the first arm (pointed by x_1). The problem for helices (which can be seen as two related substrings) is more complex than for `composition` since the two strings are initially unknown. This makes it impossible to use string matching algorithms relying on a preprocessing of the string searched. The most naive approach that successively tries all possible positions for the first and second string is obviously quadratic. However, in our case, the distance between the regions where the words may appear is constrained by the length parameters b_{min} and b_{max}. Together with parameters l_{min} and l_{max}, this makes the complexity of the naive approach linear in the text length. This is therefore the method implemented. A similar approach can be used for other bounds.
- `duplex`[...](x_1, x_2, y_3, y_4): this constraint differs from the previous one by the precise fact that there is no

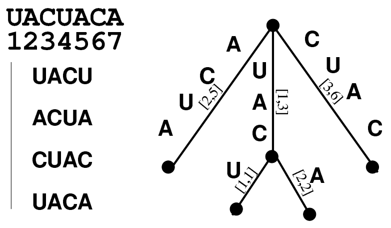


Figure 6: The k factor tree (with $k = 4$) for *UACUACA*. This data structure represents the set of substrings of length 4 of the text.

possible b_{min} and b_{max} parameters since the two interacting substrings do not necessarily appear on the same sequence. The previous naive approach is therefore impractical. We decided to use a specialized version of suffix-trees [McCreight, 1976; Ukkonen, 1992] that captures occurrences of patterns of bounded length. This data structure, called a *k-factor tree* [Allali and Sagot, 2003] allows to perform string search in time linear in the length of the pattern searched (independently of the text length). The data structure, illustrated in Fig. 6, is built once before the search, in space and time linear in the length of the text [McCreight, 1976; Ukkonen, 1992]. The associated filtering algorithm does not enforce generalized bound arc consistency but is only triggered when one of the two variables x_1 or y_3 is assigned. *All* the occurrences of the Watson-Crick reverse complement can then be efficiently found in the k-factor tree and used to update the bounds of the other variables (the position of the first and last possible occurrences define the new bounds).

Because these constraint propagation are quite expensive compared to the simple `distance` constraint, and in order to avoid repeated useless applications of the filtering algorithms, once a support is found it is memorized and will not be sought again until one of its value is deleted (as in AC2001 [Bessiere and Regin, 2001]).

3 Results and discussion

This approach has been implemented in C++ and results in a specific solver called MILPAT: **M**otifs and **I**nter-mo**L**ecular motifs searching tool using cs**P** form**A**lism and solving **T**echniques. We tested our approach on different RNA gene search problems in order to assess its efficiency and modelling capacities.

3.1 tRNAs

The tRNA structure and sequence profiles are perhaps the best studied among RNAs; hence, they are very appropriate for a first benchmarking.

tRNA genes include four helices corresponding respectively to A-stem (7 nucleotide pairs), D-stem (from 3 to 4 nucleotide pairs), C-stem (5 nucleotide pairs) and T-stem (5 nucleotide pairs), six loops corresponding respectively to the single strand between A-stem and D-stem (sequence UN with U invariant), D-loop (4 to 14 nucleotides), the single strand between D-stem and C-stem (one nucleotide), C-loop (6 to 60 nucleotides), the single strand between C-stem and T-stem (also called V-loop, 2 to 22 nucleotides), T-loop (NUC).

Software (genome size)	*E. coli* (4.610^6)	*S. cereviciae* (12.0710^6)
PatScan	1 min. 32	1 h 40
RnaMotif	4 s.	8h40
RnaMot	2 min.	92 h
MILPAT (order A)	39 s	1 h52
MILPAT (order B)	39 s	20 min.

Table 1: Comparison of the time efficiency.

The signature of tRNAs used here is deliberately a simple one that can be modelled in all existing general purpose tools. We have concentrated on finding sequences that can adopt a cloverleaf-like secondary structure within given ranges of stem and loop lengths. We searched the *Escherichia coli* and *Saccharomyces cerevisiae* genomes.

We compared the time execution of MILPAT with three other general purpose programs. The tRNA signature used in our comparisons is from Gautheret and *al.* [Gautheret *et al.*, 1990]. It includes four helices constraints, 14 distance constraints and 2 composition constraints (see Fig. 4). The results of this comparison are shown in Table 1. For each genome search test, all the programs gave the same number of solutions (545 solutions are found for the *E. coli* genome and 849982 for the *S. cerevisiae* genome).

On the computing efficiency basis, three groups may be formed from the slowest to the fastest: (i) RnaMot and RnaMotif, (ii) Patscan and MILPAT with variable selection order A, and (iii) MILPAT with variable selection order B. It is well known that variable assignment order may have a significant influence on efficiency. The static order A used by MILPAT consists in ordering variables according to the topological order of the elements in the structured motif, from 5' to 3'. Order B is a dynamic order following the first fail principle: most constrained variables are chosen first by the backtrack algorithm. Without this order, MILPAT already has an execution time close to the most efficient program, PatScan. Just changing the order leads to an early pruning of the search tree and a considerably improved execution speed for *Saccharomyces cerevisiae*.

3.2 snoRNAs

To validate the ability of MILPAT to model interactions between different molecules, we performed a computational scan of the *Pyrococcus abyssi* genome for C/D snoRNA genes. Since no existing general purpose tool allows to model interaction between a snoRNA and its target, we compared MILPAT to Snoscan, a tailored software for the C/D snoRNA genes. This program sequentially identifies six specific components of these genes (see Fig. 5): a RUGAUGA string (so called C box), a sequence region, able to form a duplex with another "target" sequence and a CUGA string (so called D box). We used a *S. cerevisiae* tailored version of snoScan as no archae-bacteria version is available. This fact probably explains the limited sensitivity shown in Table 2. The descriptor

Software	Solutions	True positives	Time
SnoScan	1611	27	20 min.
MILPAT	852	42	8 s.

Table 2: *Pyrococcus abyssi* genome (1.710^6 characters) - 59 annotated snoRNAs.

used by MILPAT is described in Fig. 5. The missing annotated genes (17 out of 59) are due to the current limitation of the duplex constraint to Watson/Crick matches. These first results show the modelling flexibility and solving efficiency of MILPAT.

4 Conclusion

The main aim of our work is to offer a way of describing new generations of RNA patterns, including the specification of complexes which can be formed by anti-sense interactions between different regions of a genome. The use of CSP methodology together with efficient pattern matching data structures and algorithms provides increased efficiency, extended modelling capabilities for intermolecular interactions and an easily extensible framework.

Beyond this ability to describe inter and intra-molecular interactions with a great flexibility, a number of evolutions are possible to improve MILPAT efficiency and modelling capabilities, including the ability to describe optional or alternative motifs. Within the framework of biological applications, these possibilities are essential to be closer to the structural reality of the molecules.

In its current version, MILPAT is just providing all the true occurrences (satisfying all constraints). It does not optimize any scoring system based on mismatches, thermodynamics or probabilistic parameters. Taking into account such information would require the use of more complex Weighted CSP algorithms such as in [Schiex *et al.*, 1995; Larrosa and Schiex, 2004].

References

[Allali and Sagot, 2003] J. Allali and M. F. Sagot. The at most k-deep factor tree. *Theory of Computer Science*, 2003. in submission.

[Altman *et al.*, 1994] RB Altman, B Weiser, and HF Noller. Constraint satisfaction techniques for modeling large complexes: Application to the central domain of 16s ribosomal rna. In *Proceedings of the second international conference on Intelligent Systems for Molecular Biology*, pages 10–18, 1994.

[Baeza-Yaltes and Gonnet, 1992] R Baeza-Yaltes and GH Gonnet. A new approach to text searching. In *Communications of the ACM*, volume 35, pages 74–82, 1992.

[Barahona and Krippahl, 1999] P Barahona and L Krippahl. Applying constraint programming to protein structure determination. *CP*, pages 289–302, 1999.

[Bessiere and Regin, 2001] C Bessiere and C Regin, J. Refining the basic constraint propagation algorithm. In *IJCAI*, pages 309–315, 2001.

[Billoud *et al.*, 1996] B Billoud, M Kontic, and A Viari. Palingol: a declarative programming language to describe nucleic acids' secondary structures and to scan sequence database. *Nucleic Acids Res*, 24(8):1395–403, 1996.

[Bockhorst and Craven, 2001] J Bockhorst and M Craven. Refining the structure of a stochastic context-free grammar. In *IJCAI*, pages 1315–1322, 2001.

[Dechter, 2003] R Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

[Dsouza *et al.*, 1997] M Dsouza, N Larsen, and R Overbeek. Searching for patterns in genomic data. *Trends Genet*, 13(12):497–8, 1997.

[Eddy and Durbin, 1994] SR Eddy and R Durbin. Rna sequence analysis using covariance models. *Nucleic Acids Res*, 22(11):2079–88, 1994.

[Eddy, 1996] SR Eddy. Rnabob: a program to search for rna secondary structure motifs in sequence databases. Manual, 1996.

[Eidhammer *et al.*, 2001] I Eidhammer, D Gilbert, I Jonassen, and M Ratnayake. A constraint based structure description language for biosequences. *Constraints*, 6:173–200, 2001.

[Gaspin and Westhof, 1994] C Gaspin and E Westhof. The determination of the secondary structures of RNA as a constraint satisfaction problem. In Frontiers in Artificial Intelligence and Applications. IOS Press, editors, *Advances in Molecular Bioinformatics*. S. Schulze-Kremer, 1994.

[Gautheret *et al.*, 1990] D Gautheret, F Major, and R Cedergren. Pattern searching/alignment with RNA primary and secondary structures: an effective descriptor for trna. *Comput Appl Biosci*, 6(4):325–31, 1990.

[Hentenryck *et al.*, 1992] V Hentenryck, P, Y Deville, and M Teng, C. A generic arc-consistency algorithm and its specializations. *Artif. Intell.*, 57(2-3):291–321, 1992.

[Larrosa and Schiex, 2004] J. Larrosa and T. Schiex. Solving Weighted CSP by maintaining Arc Consistency. *Artificial Intelligence*, 159(1-2):1-26, 2004.

[Lhomme, 1993] O Lhomme. Consistency techniques for numeric CSPs. In *the 13-th International Joint Conference on Artificial Intelligence*, pages 232–238, 1993.

[Macke *et al.*, 2001] TJ Macke, DJ Ecker, RR Gutell, D Gautheret, DA Case, and R Sampath. Rnamotif, an RNA secondary structure definition and search algorithm. *Nucleic Acids Res*, 29(22):4724–35, 2001.

[Major *et al.*, 1991] F Major, M Turcotte, D Gautheret, G Lapalme, E Fillion, and R Cedergren. The combination of symbolic and numerical computation for three-dimensional modeling of RNA. *Science*, 253:1255–1260, 1991.

[McCreight, 1976] EM McCreight. A space-economical suffix tree construction algorithm. *Journal of the ACM*, 23(2):262–272, 1976.

[Muller *et al.*, 1993] G Muller, C Gaspin, A Etienne, and E Westhof. Automatic display of RNA secondary structures. *Cabios*, 9(275):551–561, 1993.

[Policriti *et al.*, 2004] Alberto Policriti, Nicola Vitacolonna, Michele Morgante, and Andrea Zuccolo. Structured motifs search. In *Proceedings of the eighth annual international conference on Computational molecular biology*, pages 133–139. ACM Press, 2004.

[Sakakibara *et al.*, 1994] Y Sakakibara, M Brown, R Hughey, IS Mian, K jölander, RC Underwood, and D Haussler. Recent methods for rna modeling using stochastic context-free grammars. In *CPM '94: Proceedings of the 5th Annual Symposium on Combinatorial Pattern Matching*, pages 289–306. Springer-Verlag, 1994.

[Schiex *et al.*, 1995] T Schiex, H Fargier, and G Verfaillie. Valued Constraint Satisfaction Problems: hard and easy problems. In *Proc. of the International Joint Conference in AI, Montreal, Canada*, 1995.

[Smith and Waterman, 1981] TF Smith and MS Waterman. Identification of common molecular subsequences. *J Mol Biol*, 147(1), 1981.

[Storz, 2002] G Storz. An expanding universe of noncoding rnas. *Science*, 296(5571):1259, 2002.

[Ukkonen, 1992] E Ukkonen. Constructing suffix-trees online in linear time. In *Algorithms, Software, Architecture: Information Processing 92*, pages 484–492, 1992.

[Vialette, 2004] S. Vialette. On the computational complexity of 2-interval pattern matching problems. *Theor. Comput. Sci.*, 312(2-3):223–249, 2004.

[Wu and Manber, 1991] S Wu and U Manber. Fast text searching with errors. Technical report, University of Arizona, 1991.

D^3G^2A: the new Distributed Guided Genetic Algorithm for CSOPs

Bouamama Sadok and Ghedira Khaled

SOIE, ISG, University* of *Tunis
Sadok.Bouamama@ensi.rnu.tn , Ghedira.Khaled@isg.rnu.tn

Abstract

D^3G^2A is a newer distributed genetic Algorithm for CSOPs. Our approach benefits not only from autonomous dynamic multi-agent systems reducing GAs temporal complexity but also from GAs efficiency. D^3G^2A is enhanced by many newer parameters such as the Local optima detector LOD and the species types coefficientε. These latter allow not only diversification but also escaping from local optima.
The newer approach is experimented on the Radio Link Frequency Assignment Problem. The results show clearly that the new approach gives many improvements. In this paper, newer algorithms and their global dynamics are furnished, and experimental results are provided.

1 Introduction

CSP formalism consists of variables associated with domains and constraints involving subsets of these variables. A CSP solution is an instantiation of all variables with values from their respective domains. The instantiation must satisfy all constraints.
In the realms of CSP, the instantiation of a variable with a value from its domain is called a label. A simultaneous instantiation of a set of variables is called a compound label, which is a set of labels. A complete compound label is one that assigns values, from the respective domains, to all the variables in the CSP.

A CSOP is a CSP with an objective function f that maps every complete compound label to a numerical value. The goal is to find a complete compound label S such that f(S) gives an optimal value, and that no constraint is violated. CSOPs make up the framework to this paper.

CSOPs are generally NP-hard. They have been dealt with by complete or incomplete methods. The first ones, such as Branch and Bound [Tsang, 1993] are able to provide an optimal solution. Unfortunately, the combinatorial explosion thwarts this advantage. The second ones, such as Guided Genetic Algorithms (GGA) [Lau and Tsang, 1998] have the property to avoid the trap of local optima. They also sacrifice completeness for efficiency.

There is other distributed GAs known as Distributed Guided Genetic Algorithms. These approaches have been successfully applied to Max-CSP [Bouamama and Ghedira, 2003a, 2003b, 2004]. Basically these distributed approaches outperform the Centralized Genetic Algorithms (GGAs) [Lau and Tsang, 1998], which are especially known to be expensive in time. As these approaches give good results with the Max-CSPs, in terms of both optimality and quality, why not to adopt the same idea for CSOPs. This is the aim of this paper. Our interest in GAs is also motivated by their proven usefulness in many fields [Michael et al., 1999].

2 CSOP Formalism

A *constraint satisfaction and optimization problem* [Tsang, 1993] , or CSOP, is a quadruple (*X*, *D*, *C*, *f*); whose components are defined as follows:

– *X* is a finite set of variables $\{x_1, x_2, \ldots x_n\}$.
– *D* is a function which maps each variable in X to its domain of possible values, of any type, and D_{xi} is used to denote the set of objects mapped from x_i *by D*. D can be considered as $D = \{D_{x_1}, D_{x_2}, \ldots, D_{x_n}\}$;
– *C* is a finite, possibly empty, set of constraints on an arbitrary subset of variables in *X*. these constraints are represented in Extension or in Intention.
– *f* an objective function which maps every instantiation to a numerical value.

3 Dynamic Distributed Double Guided Genetic Algorithm FOR CSOP

3.1 Basic principles

Our approach draws basically on the concept of both species and ecological niches. The species consists of several organisms having common characteristics whereas the ecological niche represents the task performed by a given species. Goldberg sets that the sexual differentiation based on specialization via both the building of species and the exploitation of ecological niches provides good results [Goldberg, 1989]. A certain number of methods have been settled in order to favorite the building of ecological niches [Ghedira, 2002] in GAs.

So, the idea here is to partition the initial population into sub-populations and to assign each one of them to an agent called Species agent. A given sub-population consists of chromosomes having their fitness values in the same range. This range, said FVR, is called the specificity of the Species agent Species_{FVR}. Species agents are in interaction, in order to reach an optimal solution for the problem. For this reason, each Species agent performs its own GA. The latter is guided by both template[Lau and Tsang, 1998] concept and min-conflict heuristic[Minton, 1992]. An intermediary agent is necessary between the society of Species agents and the user, essentially to detect the best partial solution reached during the dialogue between the Species. This agent, called Interface, may also possibly create new Species agents.

3.2 Min-Conflict-Heuristic and the Template Concept

each chromosome is attached to a *template* [Tsang, 1999] that is made up of weights referred to as $\text{template}_{i,j}$. Each one of them corresponds to $\text{gene}_{i,j}$ where i refers to the chromosome and j to the position. $\delta_{i,j}$ represents the sum of costs of violated constraints by $\text{gene}_{i,j}$. These weights are updated by means of the *penalty* operator (see sub-section 3.7).

Templates will be used by GA in replacement. As we use the min-conflict-heuristic, replacement have to be elitist, i.e a chromosome is replaced by a better chromosome. For this, heavier templates genes have more probability to be replaced.

3.3 Preparing CSOP

Relationship between both genetic and CSOP formalisms is outlined as below; each chromosome (respectively gene) is equivalent to a CSOP potential solution (respectively variable). Moreover, each allele corresponds to a value.

Given an objective function *f,* we define an fitness function (FF) g which will be used by the optimization process [Lau and Tsang, 1998].

$$g(ps) = f(ps) + \lambda * \Sigma(CP_i * I_i(ps)) \qquad (1)$$

Where *ps* is a potential solution, λ is a parameter to the algorithm called Regularization parameter. It is a parameter that determines the proportion of contribution that penalties have in the fitness function.

CP_i is the penalty for gene_i (all CP_i are initialized to 0) and I_i is an indication of whether ps satisfies all constraints or not:

$$I_i(ps) = \begin{cases} 1 \text{ if ps satisfies all constraints;} \\ 0 \text{ otherwise.} \end{cases} \qquad (2)$$

Let us mention here that I_i is specific for every gene_i. for this, we sum over the index *i* the I_i values in order to express the contribution of every gene in the solution. So if a solution dos not satisfy all the problem constraints, the contribution of I_i will minimize the fitness function.

3.4 Agent Structure

Each agent has a simple structure: its acquaintances (the agents it knows and with which it can communicate), a local knowledge composed of its static and dynamic knowledge, and a mailbox where it stores the received messages to be later processed one by one.

Species Agent

A Specie agent has got as acquaintances the other Specie agents and the Interface agent. Its static knowledge consists of the CSOP data (i.e. the variables, their domains of values, the constraints and the objective function), the specificity (i.e. the fitness function range) and its local GA parameters (mutation probability, cross-over probability, number of generations, etc.). Its dynamic knowledge takes components as the population pool, which varies from one generation to another (chromosomes, population size).

Interface Agent

An Interface agent has as acquaintances all the Specie agents. Its static knowledge consists of the ∑CSP data. Its dynamic knowledge includes the best chromosome (i.e. the chromosome having the best fitness function value).

3.5 Global Dynamic

The Interface agent randomly generates the initial population and then partitions it into sub-populations accordingly to their specificities i.e. the fitness value range FVR. After that the former creates Species agents to which it assigns the corresponding sub-populations. Then the Interface agent asks these Species to perform their optimization processes. So, before starting its own optimization process, i.e. its own behaviour, each Specie agent, Species_{FVR}, initializes all templates and penalties counters corresponding to its chromosomes. After that it carries out its genetic process on its initial sub-population, i.e. the sub-population that the Interface agent has associated to it at the beginning. This process, which will be detailed in the algorithms, returns a sub-population "pop" that has been submitted to the crossing and mutating steps only once, i.e. corresponding to one generation. For each chromosome of pop, Specie_{FVR} computes their fitness function values FV according to formula (1). Consequently, two cases may occur. The first one corresponds to a chromosome having an FV in the same range as its parents. In this case, the chromosome replaces one of the latter randomly chosen. In the second case, this value (FV) is not in the same range (FVR), i.e, the specificity of the corresponding Species_{FVR}. Then the chromosome is sent to another Species_{FV} if such agent already exists, otherwise it is sent to the Interface agent. The latter creates a new agent having FV as specificity and transmits the quoted chromosome to it. Whenever a new Species agent is created, the Interface agent informs all the other agents about this creation and then asks the new Species to perform its optimization process. Note that message processing is given a priority. So, whenever an agent receives a message, it stops its behaviour, saves the context, updates its local knowledge, and restores the context before resuming its behaviour.

Here we describe the syntax used in the Figures:

- sendMsg (sender, receiver,'message'): 'message' is sent by "sender" to "receiver".
- getMsg (mailBox): retrieves the first message in mailBox.

```
Assessing-Message
1. m ← getMsg (mailBox)
2. Case (m) in
3.  optimization-process (sub-population):
            apply-behavior (sub-population)
4. take-into-account (chromosome):
            population-pool ← population-pool ∪ {chromosome}
5.  inform-new-agent (Specie_FV): list-acquaintances ← list-acquaintances ∪ {Specie_FV}
6.  stop-process: stop-behaviour
```

Figure 1: Message processing relative to Specie_{FVR}

```
Apply-behavior (initial-population)
1.  init-local-knowledge
2.  for k := 1 to number-of-generations do
3.      template-updating (initial-population)
4.      pop ← genetic-process (initial-population)
5.      best-FV ← 0
6.      for each chromosome_j in pop do
7.          FV_j ← compute-augmented-fitness-value (chromosome)
8.          if best-FV ≤ FV_j
9.            then best-FV ← FV_j
10.                clear (LO-chromosomes-list)
11.                LO-chromosomes-list ← LO-chromosomes-list ∪ {chromosome_j}
12.         if (FV_j ∈ range_i)
13.           then  replace-by (chromosome_j)
14.           else  if exist-agent (Specie_FVR)
15.                   then   sendMsg (Species_i, Specie_FVR, 'take-into-account (chromosome_j)')
16.                   else   sendMsg (Specie_FVR, Interface, 'create-agent (chromosome_j)')
17.     end for
18.     if best-FV= last-FV
19.         then stat-counter ← stat counter + 1
20.         else last-FV ← best-FV
21.     if stat-counter= LOD_i
22.         then last-stat-FV ← last-FV
23.              penalize(LO-chromosomes-list)
24. end for
25. sendMsg (Specie_FV, Interface,' result (one-chromosome, specificity)' )
```

Figure 2: Behaviour relative to $\mathrm{Species}_{FVR}$

```
Genetic process
1.  mating-pool ← matching (population-pool)
2.  template-updating (mating-pool)
3.  offspring-pool-crossed ← crossing (mating-pool)
4.  offspring-pool-mutated ← mutating (offspring-pool- crossed)
5.  return offspring-pool-mutated
```

Figure 3: The Genetic process

3.6 Guided Cross-over and Guided Mutation

Out of each pair of chromosomes, the cross-over operator produces a new child as described in Figures 4 and 5. The child inherits the best genes, i.e. the "lighter" ones, from its parents. The probability, for a parent $\mathrm{chromosome}_i$ ($i=i_1$ or i_2), where $sum = template_{i1,j} + template_{i2,j}$ to propagate its $\mathrm{gene}_{i,j}$ to its child chromosome is equal to $1\text{-}template_{i,j} / sum$. This confirms the fact that the "lighter" genes, i.e. having the best FV, are more likely than the other to be passed to the child.

For each one of its chromosomes selected according to the mutation probability P_{mut}, $\mathrm{Species}_{FVR}$ uses the min-conflict-heuristic first to determine the gene (variable) involved in the worst FV, secondly to select from this gene domain the value that violates the minimal number of constraints and finally to instantiate this gene with this value (see Figure 6).

```
Cross-over (chromosome_i1, chromosome_i2)
1.for j :=1 to size (chromosome_i1) do
2..   sum ← template_i1,j + template_i2,j
3.  if (random-integer [0, sum – 1]< template_i1,j )
4.  then gene_i3,j ← gene_i2,j
5.  else gene_i3,j ← gene_i1,j
6. Return chromosome_i3
```

Figure 4: Cross-over operator

```
Crossing (mating-pool)
1.    if (mating-pool size < 2)
2.    then return mating-pool
3.    for each pair in mating-pool do
4.      if (random [0,1] < Pcross)
5.      then offspring ← cross-over (first-pair, second-pair)
6.            FV ← compute-fitness-value (offspring)
7.     offspring-pool ← offspring-pool ∪ {offspring}
8.    return offspring-pool
```

Figure 5: Crossing process relative to $\mathrm{Species}_{FVR}$

```
Min-conflict-heuristic (chromosome_i)
1. δ_i,j ← max (template_i) /*δ_i,j is associated to gene_i,j which is in turn associ-
       ated to the variable v_j*/
2.   FV* ← 0
3.     for each value in domain of v_j do
4.        FV← compute-fitness-value (value)
5.        if (FV > FV*)
6.      then   FV* ← FV
7.      value* ← value
8.   value (gene_i,j) ← value*
9.   update (template_i)
10. return FV*
```

Figure 6: Min-conflict-heuristic relative to $\mathrm{chromosome}_i$

If all the Species agents did not meet any better chromosome at the end of their behaviour or they attain the stopping criterion, they successively transmit one of their randomly chosen chromosomes, linked to its specificity to the Interface agent. The latter determines and displays the best chromosome namely the one which have the best FV.

3.7 Penalty Operator and Local Optima Detector

To enhance the approach, the agents are given more autonomy and more dynamicity. In fact, we add an other GA's parameter that we call LOD for local optima detector. The latter represents the number of generation in which the neighboring does not give improvement, i.e. if the FV of the best chromosome remains unchanged for a specific number of generations; and so we can conclude that the agent optimization sub-process is trapped in a local optimum. In fact if the unchanged FV is lesser than the last stationary FV then automatically LOD have to be equal to one. Otherwise the LOD will remain unchanged i.e. LOD is a parameter to the whole optimization process and it will be dynamically updated by every agent.

Let us mention that every $\text{Specie}_{\text{FVR}}$ have to save its best FV for the next generations. This will be very useful not only in the case of stationary fitness values, but also to select the best chromosome in the species. In fact, if the best FV remain unchanged for LOD_i generation, the process will be considered as trapped in a local optimum. Thus for all the chromosomes having this FV, the related penalty counter PC_i of all its genes is incremented by one.

As we are in an optimization case, every $\text{Specie}_{\text{AFR}}$ have to send its best chromosome to the Interface Agent. The latter updates its local knowledge by this information. This must be done once after every generation. The Interface Agent will, at every attempt, compare the best chromosome he has with the best one sent by the species agents. Only those having the best FV will be maintained.

When the optimization process settles on a local optimum, the penalty of potential solution associated to this local optimum is increased. This helps the search process to escape from local optima, and drives it towards other candidate solutions. It is worth pointing out that a slight variation in the way that penalties are managed could make all the difference to the effectiveness of our approach. This is done by incrementing its penalty value by 1:

$$CP_i = CP_i + 1 \qquad (3)$$

3.8 Mutation and Guidance Probability

The approach, as decribed until now, can not be considered as a classic GA. In fact, in classic GAs the mutation aims to diversify a considered population and then to avoid the population degeneration [Goldberg, 1989]. In this approach, mutation operator is used improperly since it is considered as a betterment operator of the considered chromosome. However, if a gene value was inexistent in the population there is no way to obtain it by cross-over process. Thus, it is sometimes necessary to have, a random mutation in order to generate the possibly missing gene values. Our approach is a local search method. The first known improvement mechanism of local search is the diversification of the search process in order to escape from local optima [Schiex, 1995]. No doubt, the simplest mechanism to diversify the search is to consider a noise part during the process. Otherwise the search process executes a random movement with probability p and follows the normal process with a probability $1-p$ [Schiex, 1995].

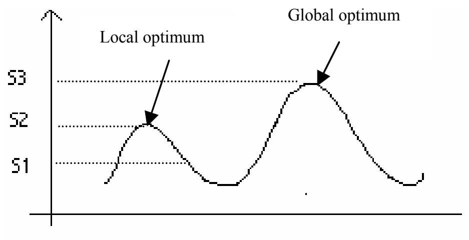


Figure 7: An example of attraction basin of local optima

In Figure 7, an example of local optima attraction basin is introduced; in a maximization case, *S2,* which is a local maximum, is better than *S1*. The passing through *S1* from *S2*, is considered as a solution destruction but give more chance to the search process to reach *S3*, the global optimum.

For all these reasons, the new proposed approach is enhanced by a random providing operator which we call guidance probability P_{guid}. Thus the approach will possess (in addition to the cross-over and mutation operators, to the generation number and to the initial-population size) a guidance operator.

The mutating sub-process will change; for each selected chromosome following mutation probability P_{mut}, the mutation will be random with a probability $1-P_{guid}$ and guided with a probability P_{guid} (Figure 8 line 3). So that in the proposed mutating sub-process it's possible to destroy a given solution in order to enhance exploration. This process is illustrated in Figures 8,9 and10.

```
Mutating (offspring-pool)
1. for each chromosome in offspring-pool do
2.    if (random [0,1]< $Pmut_i$) then if  (random [0,1]< $P_{guid}$)
3.     then Guided_Mutation   (chromosome $_i$)
4.                    else  Random_Mutation(chromosome$_i$)
5.  FV* ← compute-augmented-fitness-value(chromosome $_i$)
6.  offspring-pool-mutated ← offspring-pool-mutated ∪{$chromosome_i$}
7.  return offspring-pool-mutated
```

Figure 8: Mutating process relative to $\text{Species}_{\text{FVR}}$

```
Random_Mutation   ($chromosome_i$)
1. Choose randomly a $gene_{i,j}$
2. Choose randomly a value $v_i$ in domain of $gene_{i,j}$
3. value($gene_{i,j}$) ← $v_i$
4. Return $chromosome_i$
```

Figure 9: Random Mutation relative to chromosome_i

```
Guided_Mutation   ($chromosome_i$)
1. min-conflict-heuristic ($chromosome_i$)
2. Return $chromosome_i$
```

Figure 10: Guided Mutation relative to chromosome_i

3.9 Dynamic Approach

The main interest of the second improvement is based on the NEO-DARWINISM theory [Darwin, 1859] and on the laws of nature « *The Preservation of favoured races in the struggle for life* ». This phenomenon can be described, in nature, by an animal society in which the strongest members are luckier to be multiplied (so their crossing–over probability is high). The power of these elements allows them not to be infected by illnesses (mutation is then at a lower rate). On the contrary case the weakest limbs of these animals are frequently ill or unable to combat illnesses (mutation is frequent), usually this kind of animals can't attract females (reproduction is limited). In fact to cross-over a strong species and to give more mutation possibility for a weak species can be very worthy.

So, from now on P_{cross} and P_{mut} will be function of fitness and of a newer operator called ε. This operator is a weight having values from 0 to 1.

```
Genetic process
 1. mating-pool ← matching (population-pool)
 2. (Pcrossi, Pmuti, LODi) ← count_operator (Pcross, Pmut, LOD)
 3. template-updating (mating-pool)
 4. offspring-pool-crossed ← crossing (mating-pool)
 5. offspring-pool-mutated ← mutating (offspring-pool- crossed)
 6. return offspring-pool-mutated
```

Figure 11: The new Genetic process

In the newer optimization process, described in Figures 11 and 12, each species agent proceeds with its own genetic process. Indeed before starting the optimization process agents have to count their parameters P_{cross} and P_{mut} on the basis of their fitness values. For a given Species agent three cases are possible as described by the new genetic process detailed in figures 11 and 12.

```
Count_operator (Pcross, Pmut, LOD)
1.  if FV < (max-attained-FV / 2) then
2.          $Pcross_i \leftarrow Pcross / \varepsilon$
3.          $Pmut_i \leftarrow Pmut * \varepsilon$
4.          $LOD_i \leftarrow LOD / \varepsilon$
5.  if FV > (max-attained-FV / 2) then
6.          $Pcross_i \leftarrow Pcross * \varepsilon$
7.          $Pmut_i \leftarrow Pmut / \varepsilon$
8.          $LOD_i \leftarrow LOD * \varepsilon$
9.  if FV = (max-attained-FV / 2) then
10.         $Pcross_i \leftarrow Pcross$
11.         $Pmut_i \leftarrow Pmut$
12.         $LOD_i \leftarrow LOD$
13. if FV ≤ last-stat-FV
14.         then $LOD_i \leftarrow 1$
15. return ($Pcross_i$, $Pmut_i$, $LOD_i$)
```

Figure 12: The operator count process

4 The Radio Link Frequency Assignment Problems (RLFAP)

The French "Centre d'Électronique de l'Armement" (CELAR) has made available, in the framework of the European project EUCLID CALMA (Combinatorial Algorithms for Military Applications) set of Radio Link Frequency Assignment benchmark problems (RLFAP)[1] build from a real network, with simplified data. These benchmarks had been previously designed by the CELAR to assess several different Constraint Programming languages.

These benchmarks are extremely valuable as benchmarks for the CSP community and more largely for constraint programming:

- The constraints are all binary (involving no more than two variables), non linear and the variables have finite domains.

- These are real-world size problems, the larger instances having round one thousand variables and more than five thousand constraints. All these instances have been built from a unique real instance with 916 links and 5744 constraints in 11 connected components [Bertrand et al. 1999].

The Radio Link frequency Assignment Problem consists in assigning frequencies to a set of radio links defined between pairs of sites in order to avoid interferences. Each radio link is represented by a variable whose domain is the set of all frequencies that are available for this link. The essential constraints involve two variables F_1 and F_2:

$$|F_1 - F_2| > k_{12}$$

The two variables represent two radio links which are "close" one to the other. The constant k_{12} depends on the position of the two links and also on the physical environment. It is obtained using a mathematical model of electromagnetic waves propagation which is still very "rough". Therefore, the constants k_{12} are not necessarily correct (it is possible that the minimum difference in frequency between F_1 and F_2 that does not yield interferences is actually different from k_{12}). In practice, k_{12} is often overestimated in order to effectively guarantee the absence of interference. For each pair of sites, two frequencies must be assigned: one for the communications from *A* to *B*, the other one for the communications from *B* to *A*. In the case of the CELAR instances, a technological constraint appears: the distance in frequency between the *A->B* link and the *B->A* link must be exactly equal to 238. In all CELAR instances, these pairs of links are represented as pairs of variables numbered *2k* and *2k+1*. The possibility of expressing constraints such as $|F_1 - F_2| > k_{12}$ suffices to express the graph coloring problem and it is therefore clear that the RLFAP is NP-hard [Lau and Tsang, 1998].

Let us mention here that we will take the same formulation as that found in [Lau and Tsang, 1998]. In this case we will consider only the solvable instances. i.e the instances 1,2,3,4,5 and 11. In fact only the these instances could be considered as CSOP, as they have no constraint violation.

[1] RLFAPis available in Centre d'Electronique et de l'Armement (France), via http://www.inra.fr/bia/T/schiex/Doc/CELAR.shtml

5 Experimentation

5.1 Experimental design

The goal of our experimentation is to compare a distributed implementation with a centralized one of genetic algorithm enriched by both template concept and min-conflict-heuristic. The first implementation is referred to as Distributed Guided Genetic Algorithm (D^3G^2A) whereas the second one as Guided Genetic Algorithm (GGA). The implementation has been done with ACTALK [BRI 89], a concurrent object language implemented above the Object Oriented language SMALLTALK-80. This choice of Actalk is justified by its convivial aspect, its reflexive character, the possibility of carrying out functional programming as well as object oriented programming , and the simulation of parallelism.

In our experiments we carry out 30 times the algorithms and we take the average without considering outliers. Concerning the GA parameters, all the experimentations employ a number of generations (NG) equal to 10, a size of initial population equal to 1000, a cross-over probability equal to 0,5, a mutation probability equal to 0,2, a probability of guidance equal to 0.5, LOD is equal to 3, λ equal to 10 and ε is equal to 0.6.

The performance is assessed by the two following measures:

- Run time: the CPU time requested for solving a problem instance,
- Fitnes function value: the solution guiven by the algorithm.

The first one shows the complexity whereas the second recalls the quality. In order to have a quick and clear comparison of the relative performance of the two approaches.

Experimental results

In order to have a quick and clear comparison of the relative performance of the two approaches, we compute ratios of GGA and D^3G^2A performance using the Run time and the fitness value as follows:

- CPU time Ratio=GGA CPU time / D^3G^2A CPU time.
- Fitness ratio= FV of D^3G^2A / FV of GGA

Thus, GGA performance is the numerator when measuring the CPU time ratios, and the denominator when measuring fitness value ratio. Then, any number greater than 1 indicates superior performance by D^3G^2A.

Let us remember that we report only the solvable instances of the RFLAP. For evry one of the latters exepriments are repeatedly performed. The average is then presented.

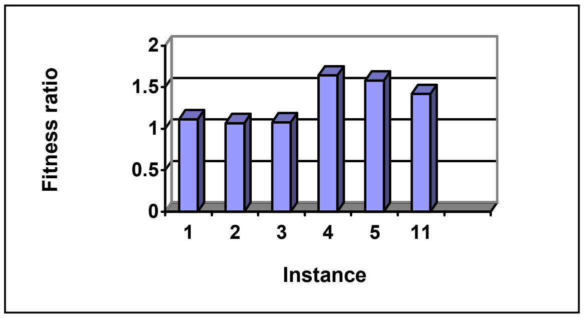


Figure 13: Fitness ratios

From the solution quality point of view shown in figure 13, the D^3G^2A always finds better solution than GGA or same one. This ratio is more significant for instances 4,5 and 11(see peaks in figure 13). The fitness ratios average is 1.31646. This is the result of the diversification and the intensification used in our approach.

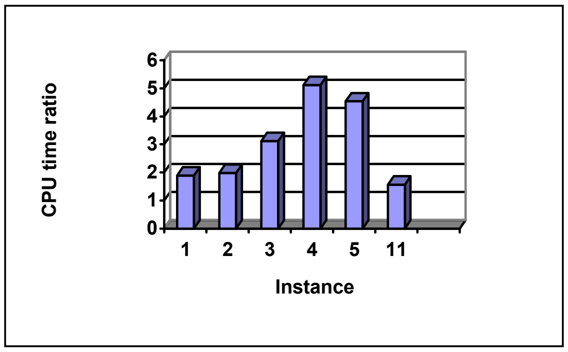


Figure 14: CPU time ratios

Form the CPU time point of view (figure. 14), D^3G^2A requires less time for all the instances of the problem. It requires until 5.122 less time for the same example. In average the CPU time ratio is equal to 3.035.

We have come to these results thanks to agent interaction reducing GA temporal complexity. In fact, the comunication between agents helps them to in the solution investigation. The CPU time is in the other hand reduced thanks to the new proposed genetic process. In the latter, both diversification (by random mutations and by LOD) and guidance are used. The first one helps the optimization process to escape from local optima. The second one, intensifies the search helping it to attain, rapidly, better fitness function values.

6 Conclusion And Perspectives

We have developed a newer approach called D^3G^2A. This approach is a dynamic distributed double guided genetic algorithm enhanced by three new parameters called guidance probability P_{guid}, the local optima detector LOD and the weight ε,. The latter is a weight used by Species agents to determine their own genetic process parameters on the basis of their chromosomes Fitness values. Compared to the centralized guided genetic algorithm and applied to RLFAP, our new approaches have been experimentally shown to be better in terms of fitness value and CPU time.

The improvement is due to both diversification and guidance. The first increases the algorithm convergence by escaping from local optima attraction basin. The latter helps the algorithm to attain optima. Consequently D^3G^2A gives more chance to the optimization process to visit all the search space. We have come to this conclusion thanks to the proposed mutation sub-process. The latter is sometimes random, aiming to diversify the search process, and sometimes guided in order to increase the best of the fitness fonction value.

The genetic sub-process of D^3G^2A Species agents will no longer be the same depending on their fitness values. This operation is based on the species typology. The sub-population of a species agent can be considered as strong or weak with reference to its fitness value. For a strong species, it's better to increase cross-over probability and to decrease mutation probability. However, when dealing with a weak species, cross-over probability is decreased and mutation probability is increased. The occurrence of these measures not only diversifies the search but also explore wholly its space.

No doubt further refinement of this approach would allow its performance to be improved. Further works could be focused on applying these approaches to solve other real hard CSOPs and valued CSPs [Schiex et al., 1995].

References

[Briot, 1989] Briot J. P. Actalk: a testbed for classifying and designing Actor languages in the Smalltalk-80 Environment. *Proceedings of the European Conference on Object-Oriented Programming (ECOOP'89).* British Computer Society Workshop Series Cambridge University Press. , July 1989

[Bertrand et al. 1999] B. Cabon, S. de Givry, L. Lobjois, T. Schiex, J. P. Warners. *Radio Link Frequency Assignment*, Constraints 4(1): 79-89 (1999).

[Bouamama S, Ghédira, 2003a] Bouamama S and Ghédira K. D^2G^2A and D^3G^2A: a new generation of Distributed Guided Genetic Algorithms for Max_CSPs, In *Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics (SCI'03).* Orlando,Florida, USA, 2003.

[Bouamama and Ghédira, 2003b] Bouamama S and Ghédira K. D^2G^2A: a Distributed double Guided Genetic Algorithm for Max_CSPs, In *Proceedings of the Seventh International Conference on Knowledge-Based Intelligent Information & Engineering Systems (KES'03)***.** Oxford, UK,2003.

[Bouamama S, Ghédira, 2004] Bouamama S and Ghédira K. ED^3G^2A: an Enhanced version of the Distributed Guided Genetic Algorithms for Max_CSPs, In *Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics (SCI'04).* Orlando,Florida, USA, 2004.

[Dejong and Spears, 1989] Dejong K. A. and Spears W. M. Using Genetic Algorithms to solve NP-Complete problems. *George Mason University, Fairfax, VA*, 1989.

[Darwin, 1859] Darwin C. The Origin of Species, 1859, *Sixth London Editions, 1999.*

[Ghédira and Jlifi, 2002] Ghédira K & Jlifi B. A Distributed Guided Genetic Algorithm for Max_CSPs. *Journal of sciences and technologies of information (RSTI), journal of artificial intelligence series (RIA),* volume 16 N°3/2002.

[Goldberg, 1989] Goldberg D.E. Genetic algorithms in search, Optimisation, and Machine Learning, *edition Addison-Wesley.*1989.

[Holland, 1975] Holland J. Adaptation in natural and artificial systems. *Ann Arbor: The university of Michigan Press,* 1975.

[Lau and Tsang, 1998] Lau T.L and Tsang E.P.K. Solving The Radio Link Frequency Assignment Problem With The Guided Genetic Algorithm, In *Proceedings of NATO symposium on radio length frequency assignment staring and conservation systems , UK.* Albrg, Denmark, october 1998.

[Michael et al., 1999] Michael B., Frank M. and Yi P. Improved Multiprocesor Task scheduling Using Genetic Algorithms, In *Proceedings of the twelfth International Florida AI Research Society Conference FLAIRS' 99,* AAAI press, p. 140-146, 3-5 May 1999.

[Schiex et al., 1995] Schiex T., Fargier H. and Verfaillie G., Valued constrained satisfaction problems: hard and easy problems. In *proceeding of the 14th IJCAI, Montreal, Canada*. august 1995.

[Tsang, 1993] Tsang E.P.K. Foundations of Constraints Sat isfaction. *Academic Press Limited,*1993

[Tiourine et al.1995] Tiourine S. Hurkens C. and Lenstra J. An overview of algorithmic approaches to frequency assignment problems. *CALMA Symposium Scheveningen.* 1995

A Generalization of Generalized Arc Consistency: From Constraint Satisfaction to Constraint-Based Inference

Le Chang and **Alan K. Mackworth**
Department of Computer Science, University of British Columbia
2366 Main Mall, Vancouver, B.C. Canada V6T 1Z4
{lechang, mack}@cs.ubc.ca

Abstract

Arc consistency and generalized arc consistency are two of the most important local consistency techniques for binary and non-binary classic constraint satisfaction problems (CSPs). Based on the Semiring CSP and Valued CSP frameworks, arc consistency has also been extended to handle soft constraint satisfaction problems such as fuzzy CSP, probabilistic CSP, max CSP, and weighted CSP. This extension is based on an idempotent or strictly monotonic constraint combination operator. In this paper, we present a weaker condition for applying the generalized arc consistency approach to constraint-based inference problems other than classic and soft CSPs. These problems, including probability inference and maximal likelihood decoding, can be processed using generalized arc consistency enforcing approaches. We also show that, given an additional monotonic condition on the corresponding semiring structure, some of constraint-based inference problems can be approximately preprocessed using generalized arc consistency algorithms.

1 Introduction

The notion of local consistency plays a central role in constraint satisfaction. Given a constraint satisfaction problem (CSP), local consistency can be characterized as deriving new constraints based on local information. The derived constraints simplify the representation of the original CSP without the loss of solutions. Among the family of local consistency enforcing algorithms or filtering algorithms, arc consistency [Mackworth., 1977a] is one of the most important techniques for binary classic CSP. It is straightforward to extend it as generalized arc consistency [Mackworth, 1977b; Mohr and Masini, 1988] to handle non-binary classic CSPs.

To represent over-constrained and preference-based problems in the real world, researchers in the constraint processing community are increasingly interested in so-called soft constraint satisfaction problems. Fuzzy CSP, probabilistic CSP, max CSP, and weighted CSP have been proposed to address these requirements. Semiring CSP [Bistarelli *et al.*, 1997] and Valued CSP [Schiex *et al.*, 1995] are two of the most widely studied generalized frameworks. Based on the two frameworks, arc consistency is also extended as soft arc consistency to handle soft constraints [Schiex, 2000; Cooper and Schiex, 2004; Bistarelli, 2004]. The soundness and completeness of soft arc consistency, within the Semiring CSP framework, relies on the idempotency of the constraint combination operator. Moreover, the c-semiring used in the Semiring CSP framework has the special requirement of idempotency of the additive operator. The Valued CSP framework extends soft arc consistency in the Semiring CSP framework. Soft arc consistency in Valued CSP depends on the strictly monotonic constraint combination operator or the fair valuation structure. For most soft constraint proposals, the success of soft arc consistency in the Semiring CSP framework and the Valued CSP framework has been proven [Schiex, 2000; Cooper and Schiex, 2004; Bistarelli, 2004]. For problems from other fields that cannot been as optimization problems, their representations in the Semiring CSP and Valued CSP frameworks may not be so straightforward. Preprocessing may be needed before applying the soft arc consistency enforcing approaches to solve these problems.

Given the representation analogues of constraint-based inference (CBI) problems, including probabilistic inferences, decision-making under uncertainty, constraint satisfaction problems, propositional satisfiability, decoding problems, and possibility inferences, we present in this paper a weaker condition for applying local consistency approaches to general constraint-based inference problems based on the commutative semiring structure. The weaker condition proposed here depends only on the existence and property of the combination absorbing element and does not depend on other semiring properties. More specifically, we reduce a CBI problem to its underlying classic CSP [Cooper and Schiex, 2004] according to the weaker condition. All traditional arc consistency techniques, the most widely studied local consistency approaches, then can be applied without modification.

We also show that, by satisfying an additional monotonic condition on the semiring structure characterizing the problem, generalized arc consistency can also be used as an approximate local consistency enforcing technique for CBI problems. Here we use a user-controlled threshold value to approximate the combination absorbing element. A similar approach can be found in [Rina and David, 2001;

Bistarelli *et al.*, 2002; de Givry *et al.*, 1997].

2 Background

There are two essential operators in real world CBI problems: (1) combination, which corresponds to an aggregation of constraints, and (2) marginalization, which corresponds to focusing a specified constraint to a narrower scope. These two operators allow us to use algebraic structures to generalize CBI problem representations. More specifically, both the abstract CBI representation framework and the generalized arc consistency approach in this paper are based on the semiring structure, an important notion in abstract algebra. This section introduces the definition of a semiring and related properties.

Definition 1 (Semiring) *Let $\mathbf{A}$ be a set. Let $\oplus$ and $\otimes$ be two closed binary operators defined on $\mathbf{A}$. Here we define operator $\otimes$ as taking precedence over operator $\oplus$. $\mathbf{S} = \langle \mathbf{A}, \oplus, \otimes \rangle$ is a semiring if the operators satisfy the following axioms:*

- *Additive associativity: $\forall a, b, c \in \mathbf{A}$, $(a \oplus b) \oplus c = a \oplus (b \oplus c)$;*
- *Additive commutativity: $\forall a, b \in \mathbf{A}$, $a \oplus b = b \oplus a$;*
- *Multiplicative associativity: $\forall a, b, c \in \mathbf{A}$, $(a \otimes b) \otimes c = a \otimes (b \otimes c)$;*
- *Left and right distributivity: $\forall a, b, c \in \mathbf{A}$, $a \otimes (b \oplus c) = a \otimes b \oplus a \otimes c$ and $(b \oplus c) \otimes a = b \otimes a \oplus c \otimes a$.*

To capture the computational properties of various inference approaches, we use commutative semiring, an extended algebraic notion of semiring, to formally represent CBI problems in this paper.

Definition 2 (Commutative Semiring) *A commutative semiring $\mathbf{S} = \langle \mathbf{A}, \oplus, \otimes \rangle$ is a semiring that satisfies the following additional conditions:*

- *Multiplicative commutativity: $\forall a, b \in \mathbf{A}$, $a \otimes b = b \otimes a$;*
- *Multiplicative identity: there exists a multiplicative identity element $\mathbf{1} \in \mathbf{A}$, such that $a \otimes \mathbf{1} = \mathbf{1} \otimes a = a$ for any $a \in \mathbf{A}$;*
- *Additive identity: there exists an additive identity element $\mathbf{0} \in \mathbf{A}$, such that $a \oplus \mathbf{0} = \mathbf{0} \oplus a = a$ for any $a \in \mathbf{A}$;*

We will show in the following sections that the application of local consistency techniques depend on the existence of a multiplicative (or combination) absorbing element. It is easy to show the uniqueness of the multiplicative absorbing element given the multiplicative commutativity of a commutative semiring, according the definition below.

Definition 3 (Multiplicative Absorbing Element) *An element $\alpha_{\otimes} \in \mathbf{A}$ is the multiplicative absorbing element of a commutative semiring $\mathbf{S} = \langle \mathbf{A}, \oplus, \otimes \rangle$ if $a \otimes \alpha_{\otimes} = \alpha_{\otimes} \otimes a = \alpha_{\otimes}$ for any element $a \in \mathbf{A}$.*

Similarly the additive absorbing element $\alpha_{\oplus}$ is defined as:

Definition 4 (Additive Absorbing Element) *An element $\alpha_{\oplus} \in \mathbf{A}$ is the additive absorbing element of a semiring $\mathbf{S} = \langle \mathbf{A}, \oplus, \otimes \rangle$ if $a \oplus \alpha_{\oplus} = \alpha_{\oplus} \oplus a = \alpha_{\oplus}$ for any element $a \in \mathbf{A}$.*

Furthermore, we say that $\oplus$ is idempotent if $a \oplus a = a$, and $\otimes$ is idempotent if $a \otimes a = a$. For some semirings, we can define a partial order over the elements of S if $\oplus$ is idempotent.

Definition 5 (Partial Order $\leq_{\mathbf{S}}$ [Bistarelli, 2004]) *Given a semiring $\mathbf{S} = \langle \mathbf{A}, \oplus, \otimes \rangle$, there exist a partial order $\leq_{\mathbf{S}}$ over S such that $a \leq_{\mathbf{S}} b$, $\forall a, b \in \mathbf{A}$ if:*

- *$\oplus$ is idempotent;*
- *$a \oplus b = b$.*

Given a partial order $\leq_{\mathbf{S}}$ of semiring $\mathbf{S} = \langle \mathbf{A}, \oplus, \otimes \rangle$, we know that the additive identity element $\mathbf{0}$ is the minimum element of the ordering. In other words, $\mathbf{0} \leq_{\mathbf{S}} a, \forall a \in \mathbf{A}$. If the additive absorbing $\alpha_{\oplus}$ exists, it will be the maximum element of the ordering according to the partial order definition. Also note that the two conditions are only sufficient conditions for the existence of a partial order. For example, the commutative semiring $\mathbf{S_{prob}} = \langle \mathbb{R}^{+} \cup \{0\}, +, \times \rangle$ has a partial order while does not satisfy the two conditions.

Finally, we define two more important properties for some commutative semirings. The two properties are the foundation of applying local consistency techniques to general CBI problems.

Definition 6 (Eliminative Commutative Semiring) *A commutative semiring $\mathbf{S} = \langle \mathbf{A}, \oplus, \otimes \rangle$ is eliminative if:*

- *There exists the multiplicative absorbing element $\alpha_{\otimes} \in \mathbf{A}$;*
- *$\alpha_{\otimes} = \mathbf{0}$, in other words, the multiplicative absorbing element is equal to the additive identity element.*

Definition 7 (Monotonic Commutative Semiring) *A commutative semiring $\mathbf{S} = \langle \mathbf{A}, \oplus, \otimes \rangle$ is monotonic if:*

- *There exists a total order $\leq_{\mathbf{S}}$ on $\mathbf{A}$;*
- *The additive identity element $\mathbf{0}$ is the minimum element w.r.t. $\leq_{\mathbf{S}}$. In other words, $\mathbf{0} \leq_{\mathbf{S}} a$, $\forall a \in \mathbf{A}$;*
- *Additive Monotonic: $a \leq_{\mathbf{S}} b$ implies $a \oplus c \leq_{\mathbf{S}} b \oplus c$, $\forall a, b, c \in \mathbf{A}$;*
- *Multiplicative Monotonic: $a \leq_{\mathbf{S}} b$ implies $a \otimes c \leq_{\mathbf{S}} b \otimes c$, $\forall a, b, c \in \mathbf{A}$.*

Table 1 displays some commutative semirings with their identity and absorbing elements and properties.

In the following sections, we use bold letters to denote sets of elements and regular letters to denote individual elements. Given a set of elements $\mathbf{X}$ and an element $Z \in \mathbf{X}$, $\mathbf{X}_{-Z}$ denotes the set of elements $\mathbf{X} \setminus \{Z\}$. Given a value assignment $\mathbf{x}$ of variable subset $\mathbf{X}$ and $\mathbf{Y} \subseteq \mathbf{X}$, $\mathbf{x}_{\downarrow \mathbf{Y}}$ denotes the value assignment projection of $\mathbf{x}$ onto the variable subset $\mathbf{Y}$.

3 A Semiring-Based Unifying Framework for CBI Problems

Constraint-Based Inference (CBI) is an umbrella term for various superficially different problems. It concerns the automatic discovery of new constraints from a set of given constraints over individual entities. New constraints reveal undiscovered properties about a set of entities. A constraint here is

No.	S	$\oplus$,**0**	$\otimes$,**1**	$\alpha_\otimes$	$\alpha_\oplus$	Eliminative	Monotonic
1	$\{true, false\}$	$\vee, false$	$\wedge, true$	$false$	$true$	Yes	No
2	$[0,1]$	$max, 0$	$min, 1$	0	1	Yes	Yes
3	$\mathbb{R}^+ \cup \{0\}$	$max, 0$	min, ∞	0	∞	Yes	Yes
4	$[0,1]$	$max, 0$	$\times, 1$	0	1	Yes	Yes
5	$\mathbb{R}^- \cup \{0\}$	$max, -\infty$	$+, 0$	$-\infty$	0	Yes	Yes
6	$\mathbb{N}^+ \cup \{0\}$	$max, 0$	$+, 0$	∞	∞	No	Yes
7	$\mathbb{R}^+ \cup \{0\}$	$+, 0$	$\times, 1$	0	∞	Yes	Yes
8	$\mathbb{R}^+ \cup \{0\}$	$max, 0$	$\times, 1$	0	∞	Yes	Yes
9	$\mathbb{N}^+ \cup \{0\}$	min, ∞	$+, 0$	∞	0	Yes	Yes
10	$\mathbb{N}^+ \cup \{0\}$	min, ∞	$\times, 1$	∞	0	Yes	Yes

Table 1: Properties of Various Commutative Semirings

seen as a function that maps possible value assignments to a specific value domain. Many practical problems from different fields can be seen as constraint-based inference problems. These problems cover a wide range of topics in computer science research, including probabilistic inferences, decision-making under uncertainty, constraint satisfaction problems (CSP), propositional satisfiability problems (SAT), decoding problems, and possibility inferences.

A CBI problem is defined in terms of a set of variables with values in finite domains and a set of constraints on these variables. We use commutative semirings to unify the representation of constraint-based inference problems from various disciplines into a single formal framework [Chang, 2005], based on the synthesis of the existing generalized representation frameworks [Bistarelli *et al.*, 1997; Schiex *et al.*, 1995; Kohlas and Shenoy, 2000] and algorithmic frameworks [Dechter, 1996; Kask *et al.*, 2003; Aji and McEliece, 2000] from different fields. Formally:

Definition 8 (Constraint-Based Inference (CBI) Problem) *A constraint-based inference (CBI) problem* $\mathbf{P}$ *is a tuple* $(\mathbf{X}, \mathbf{D}, \mathbf{S}, \mathbf{F})$ *where:*

- $\mathbf{X} = \{X_1, \cdots, X_n\}$ *is a set of variables;*
- $\mathbf{D} = \{\mathbf{D_1}, \cdots, \mathbf{D_n}\}$ *is a collection of finite domains, one for each variable;*
- $\mathbf{S} = \langle \mathbf{A}, \oplus, \otimes \rangle$ *is a commutative semiring;*
- $\mathbf{F} = \{f_1, \cdots, f_r\}$ *is a set of constraints. Each constraint is a function that maps value assignments of a subset of variables to values in* $\mathbf{A}$

More specifically, we use $Scope(f)$ to denote the subset of variables that is in the scope of the constraint f. We use $\mathbf{D}_X$ to denote the value domain of a variable X. Given a variable $X \in Scope(f)$, $Scope(f)_{-X}$ denotes the variable subset $Scope(f) \setminus \{X\}$. Then we define the two basic constraint operators as follows.

Definition 9 (The Combination of Two Constraints) *The combination of two constraints f_1 and f_2 is a new constraint $g = f_1 \otimes f_2$, where $Scope(g) = Scope(f_1) \cup Scope(f_2)$ and $g(\mathbf{w}) = f_1(\mathbf{w}_{\downarrow Scope(f_1)}) \otimes f_2(\mathbf{w}_{\downarrow Scope(f_2)})$ for every value assignment $\mathbf{w}$ of variables in $Scope(g)$.*

Definition 10 (The Marginalization of a Constraint) *The marginalization of X from a constraint f, where $X \in Scope(f)$, is a new constraint $g = \bigoplus_X f$, where $Scope(g) = Scope(f)_{-X}$ and $g(\mathbf{w}) = \bigoplus_{x_i \in \mathbf{D}_X} f(x_i, \mathbf{w})$ for every value assignment $\mathbf{w}$ of variables in $Scope(g)$.*

According to the definitions of the CBI problem and the basic constraint operators, we can define the abstract inference and allocation tasks for a CBI problem.

Definition 11 (The Inference Task for a CBI Problem) *Given a subset of variables $\mathbf{Z} = \{Z_1, \cdots, Z_t\} \subseteq \mathbf{X}$, let $\mathbf{Y} = \mathbf{X} \setminus \mathbf{Z}$, the inference task for a CBI problem $\mathbf{P} = (\mathbf{X}, \mathbf{D}, \mathbf{S}, \mathbf{F})$ is defined as computing:*

$$g_{CBI}(\mathbf{Z}) = \bigoplus_{\mathbf{Y}} \bigotimes_{f \in \mathbf{F}} f \tag{1}$$

Given a CBI problem $\mathbf{P} = (\mathbf{X}, \mathbf{D}, \mathbf{S}, \mathbf{F})$, if $\oplus$ is idempotent, we can define the allocation task for a CBI problem.

Definition 12 (The Allocation Task for a CBI Problem) *Given a subset of variables $\mathbf{Z} = \{Z_1, \cdots, Z_t\} \subseteq \mathbf{X}$, let $\mathbf{Y} = \mathbf{X} \setminus \mathbf{Z}$, the allocation task for a CBI problem $\mathbf{P} = (\mathbf{X}, \mathbf{D}, \mathbf{S}, \mathbf{F})$ is to find a value assignment for the marginalized variables $\mathbf{Y}$, which leads to the result of the corresponding inference task $g_{CBI}(\mathbf{Z})$. Formally, we compute:*

$$\mathbf{y} = \arg \bigoplus_{\mathbf{Y}} \bigotimes_{f \in \mathbf{F}} f \tag{2}$$

where arg *is a prefix of operator $\oplus$. In other words,* $\arg \oplus$ *is an operator that returns arguments of the $\oplus$ operator. For example, when $\oplus = max$, $\arg \oplus = \arg max$ that returns a value assignment that leads to the maximal possible element in* $\mathbf{S}$.

In general, $\otimes$ is a combination operator in CBI problems that combines a set of constraints into a constraint with a larger scope; $\oplus_{\mathbf{Y}} = \oplus_{\mathbf{X} \setminus \mathbf{Z}}$ is a marginalization operator that projects a constraint over the scope $\mathbf{X}$ into its subset $\mathbf{Z}$, through enumerating all possible value assignments of $\mathbf{Y} = \mathbf{X} \setminus \mathbf{Z}$.

Many CBI problems from different disciplines can be embedded into our semiring-based unifying framework [Chang, 2005]. These problems include the decision task and allocation task of CSP and SAT, Max SAT and Max CSP, Fuzzy CSP, Weighted CSP, probability assessment, most probable

Input: A CBI problem $\mathbf{P} = (\mathbf{X}, \mathbf{D}, \mathbf{S}, \mathbf{F})$
Output: A generalized arc consistency CBI problem $\mathbf{P}' = (\mathbf{X}, \mathbf{D}', \mathbf{S}, \mathbf{F}')$

```
 1: Let $\mathbf{Q}$ be a queue of all the variable-constraint pairs $(X, f)$
 2: repeat
 3:   Pop the first variable-constraint pair $(X, f) \in \mathbf{Q}$
 4:   if REVISE(X, f) then
 5:     for each $g \in \mathbf{F}$ with $X \in Scope(g)$ do
 6:       Remove all tuples in $g$ with the value that is removed from $X$
 7:       for each $Z \in Scope(g)$ and $X \neq Z$ do
 8:         if Pair $(Z, g) \notin \mathbf{Q}$ then
 9:           $\mathbf{Q} := \mathbf{Q} \cup \{(Z, g)\}$
10:         end if
11:       end for
12:     end for
13:   end if
14: until $\mathbf{Q}$ is empty
15: Return $\mathbf{P}' := \mathbf{P}$
```

Figure 1: Generalization of Generalized Arc Consistency Algorithm GGAC($\mathbf{P}$)

explanation (MPE), dynamic Bayesian networks (DBN), possibility inference with various t-norms, and maximum likelihood decoding. In [Chang, 2005], we also generalized various systematic inference approaches, including exact and approximate variable elimination, exact and approximate junction tree and variants, and loopy message propagation, into this semiring-based unifying framework.

4 Applying Arc Consistency to CBI Problems

4.1 Arc Consistency and Eliminative Property

Here, we are particularly interested in CBI problems defined on a commutative semiring $\mathbf{S} = \langle \mathbf{A}, \oplus, \otimes \rangle$ with the eliminative property. More specifically, we propose in this paper that local consistency techniques in constraint processing can be extended to handle general CBI problems like probability inference and maximum likelihood decoding, if the corresponding commutative semiring of the problem representation is eliminative. Formally, we define the generalized arc consistency of a CBI problem as follows:

Definition 13 (A CBI Problem is GGAC) *A CBI Problem $\mathbf{P} = (\mathbf{X}, \mathbf{D}, \mathbf{S}, \mathbf{F})$ with an eliminative commutative semiring $\mathbf{S}$ is generalized arc consistent (GGAC) if: $\forall f \in \mathbf{F}$, $\forall X \in Scope(f)$, $\forall x \in \mathbf{D}_X$, $\exists \mathbf{w}$, a value assignment of variables $Scope(f)_{-X}$, s.t. $f(x, \mathbf{w}) \neq \alpha_\otimes$*

Figure 1 shows a generalized version of generalized arc consistency (GGAC) enforcing algorithm for a CBI problem $\mathbf{P} = (\mathbf{X}, \mathbf{D}, \mathbf{S}, \mathbf{F})$ with an eliminative commutative semiring $\mathbf{S}$. The procedure REVISE of GGAC is shown in Figure 2.

Theorem 1 (GGAC Enforces Generalized Arc Consistency) *Applying GGAC algorithm to a CBI problem $\mathbf{P} = (\mathbf{X}, \mathbf{D}, \mathbf{S}, \mathbf{F})$ with an eliminative commutative semiring $\mathbf{S} = \langle \mathbf{A}, \oplus, \otimes \rangle$ leads to a generalized arc consistent CBI problem $\mathbf{P}' = (\mathbf{X}, \mathbf{D}', \mathbf{S}, \mathbf{F}')$.*

Input: A variable $X \in \mathbf{X}$ and a constraint $f \in \mathbf{F}$
Output: TRUE if a value is removed from the domain of X else FALSE

```
 1: $flag := TRUE$
 2: for each $x \in \mathbf{D}_X$ do
 3:   for each value assignment $\mathbf{w}$ of $Scope(f)_{-X}$ do
 4:     if $f(x, \mathbf{w}) \neq \alpha_\otimes$ then
 5:       $flag := FALSE$
 6:       Break loop
 7:     end if
 8:   end for
 9:   if $flag$ then
10:     Remove $x$ from $\mathbf{D}_X$
11:     Return TRUE
12:   end if
13: end for
14: Return FALSE
```

Figure 2: Procedure REVISE(X, f) for Eliminating a Domain Value from a Variable According to the Local Constraint

Proof: Assume there exists a constraint $f' \in \mathbf{F}'$ and a variable $X \in Scope f'$ that lead to generalized arc inconsistency in $\mathbf{P}'$. We know the pair (X, f') must be popped from the queue sometime since X and f' are in $\mathbf{P}'$. However, the REVISE procedure ensures that every pair popped from the queue is generalized arc consistent, which contradicts the assumption. □

The equivalency of a CBI problem with an eliminative commutative semiring and the generalized arc consistency CBI problem yielded by GGAC algorithm in Figure 1 w.r.t. the results of their inference tasks is proven by Theorem 2.

Theorem 2 (Closure of GGAC) *Let $\mathbf{P} = (\mathbf{X}, \mathbf{D}, \mathbf{S}, \mathbf{F})$ be a CBI problem and the commutative semiring $\mathbf{S} = \langle \mathbf{A}, \oplus, \otimes \rangle$ is eliminative. Let $\mathbf{P}' = (\mathbf{X}, \mathbf{D}', \mathbf{S}, \mathbf{F}')$ be the CBI problem yielded by GGAC algorithm. For any subset of variables $\mathbf{Z} \subseteq \mathbf{X}$, the inference tasks for $\mathbf{P}$ and $\mathbf{P}'$ are equivalent.*

Proof: Let (X, f) be a pair that is revised by the procedure REVISE, where $x \in \mathbf{D}_X$ is removed because of generalized arc inconsistency. Consider the global constraint g of the combination of all the constraints in $\mathbf{P} = (\mathbf{X}, \mathbf{D}, \mathbf{S}, \mathbf{F})$. We have $g(\mathbf{X}) = f(\mathbf{X}_{\downarrow Scope(f)}) \otimes \bigotimes_{h \in \mathbf{F}, h \neq f} h(\mathbf{X}_{\downarrow Scope(h)})$. More specifically, for any value assignment $\mathbf{u}$ of variables $\mathbf{X}_{-X}$, we have $g(x, \mathbf{u}) = f(x, \mathbf{u}_{\downarrow Scope(f)}) \otimes \bigotimes_{h \in \mathbf{F}, h \neq f} h(\mathbf{u}_{\downarrow Scope(h)}) = \alpha_\otimes$, since $f(x, \mathbf{u}_{\downarrow Scope(f)}) = \alpha_\otimes$ is the absorbing element of the operator $\otimes$. Given $g(X = x, \mathbf{u}) = \alpha_\otimes$ is also the identity element of the operator $\oplus$, the inference task of $\mathbf{P}$ (Equation 1) is to compute:

$$\begin{aligned} g_{CBI}(\mathbf{Z}) &= \bigoplus_{\mathbf{Y}} g(X, \mathbf{X}_{-X}) \\ &= \bigoplus_{\mathbf{Y}} (g(X = x, \mathbf{X}_{-X}) \oplus g(X \neq x, \mathbf{X}_{-X})) \\ &= \bigoplus_{\mathbf{Y}} g(X \neq x, \mathbf{X}_{-X}) \end{aligned} \quad (3)$$

On the other hand, let us consider the global constraint g' of $\mathbf{P}' = (\mathbf{X}, \mathbf{D}', \mathbf{S}, \mathbf{F}')$. We have: $g'(\mathbf{X}) = \bigotimes_{f' \in \mathbf{F}'} f' = g(X \neq x, \mathbf{X}_{-X})$ according to the GGAC algorithm in Figure 1. Then it is straightforward to get $g'_{CBI}(\mathbf{Z}) = \bigoplus_{\mathbf{Y}} g'(\mathbf{X}) = g_{CBI}(\mathbf{Z})$. □

In other words, Theorem 2 shows that when we detect that there exists a constraint of a given CBI problem on an eliminative semiring structure that maps all its value assignments with a specific value to the multiplicative absorbing element, the value can be safely removed from the variable's domain. All value assignments with this value can be safely removed from any constraint with this variable in its scope, without modifying the computational result of the inference task.

Theorems 1 and 2 together imply the correctness of the GGAC algorithm.

Theorem 3 (Time Complexity of GGAC) *The worst case time complexity of the GGAC algorithm in Figure 1 is $O(r \cdot d^{k+1})$, where r is the number of constraints, d is the maximum domain size, and k is the maximum scope size of constraints.*

Proof: Basically the GGAC algorithm is a straightforward revision of the generalized arc consistency enforcing algorithm for classic non-binary CSPs [Mackworth, 1977b]. For each constraint with at most k variables in its scope we need d^k checks. Each variable-constraint pair enters the queue at most d times, so the total number of checks is $O(r \cdot d^{k+1})$. □

We extend the application of Theorem 2 by introducing another equivalency statement of the inference task for a CBI problem.

Theorem 4 *Solving the inference task of a CBI problem $\mathbf{P} = (\mathbf{X}, \mathbf{D}, \mathbf{S}, \mathbf{F})$ is equivalent to solving $\mathbf{P}' = (\mathbf{X}, \mathbf{D}, \mathbf{S}, \mathbf{F}')$, where $\mathbf{F}' = \mathbf{F_P} \cup \{f\}$, $\mathbf{F_P} \subset \mathbf{F}$ and $f = \bigotimes_{h \in \mathbf{F} \setminus \mathbf{F_P}} h$.*

Proof: Easy to prove given the definition of the inference task of a CBI problem in Equation 1. □

Combining Theorem 2 and Theorem 4 lead to the local consistency property of general CBI problems. We do not have to focus on the original constraints in $\mathbf{F}$ of the CBI problem $\mathbf{P} = (\mathbf{X}, \mathbf{D}, \mathbf{S}, \mathbf{F})$. We can combine some original constraints to a local constraint, then apply the conclusion of Theorem 2 to refine the representation of the original CBI problem.

If the allocation task can be defined on a CBI problem $\mathbf{P} = (\mathbf{X}, \mathbf{D}, \mathbf{S}, \mathbf{F})$, in other words, $\oplus$ is idempotent, we have the analogous result, as shown in Theorem 5.

Theorem 5 (Closure of GGAC for Allocation Task) *Let $\mathbf{P} = (\mathbf{X}, \mathbf{D}, \mathbf{S}, \mathbf{F})$ be a CBI problem and the commutative semiring $\mathbf{S} = \langle \mathbf{A}, \oplus, \otimes \rangle$ is eliminative. $\oplus$ is idempotent. Let $\mathbf{P}' = (\mathbf{X}, \mathbf{D}', \mathbf{S}, \mathbf{F}')$ be the CBI problem yielded by GGAC algorithm. For any subset of interested variables $\mathbf{Z} \subseteq \mathbf{X}$, we have the allocation tasks for $\mathbf{P}$ and $\mathbf{P}'$ are equivalent.*

Proof: Similar to the proof of Theorem 2. It is easy to show that the value $x \in \mathbf{D}_X$ cannot appear in any value assignment that leads to the inference task's result $g_{CBI}(\mathbf{Z})$. □

Through shrinking the domain of a variable as well as deleting possible value assignments of constraints, the size of the original CBI problem is reduced by factor $(|\mathbf{D}_X| - 1)/|\mathbf{D}_X|$. Repeatedly applying the GGAC algorithm, we will

```
Input: A CBI problem $\mathbf{P} = (\mathbf{X}, \mathbf{D}, \mathbf{S}, \mathbf{F})$ and a variable
    subset $\mathbf{Z}$ of interest
Output: $g_{CBI}(\mathbf{Z}) = \bigoplus_{\mathbf{X} \setminus \mathbf{Z}} \bigotimes_{f \in \mathbf{F}} f$
 1: $\mathbf{P} := GGAC(\mathbf{P})$
 2: Let $\mathbf{Y} = \mathbf{X} \setminus \mathbf{Z}$
 3: Choose an elimination ordering $\sigma =< Y_1, \cdots, Y_k >$ of
    $\mathbf{Y}$
 4: for $i = k$ to 1 do
 5:    $\mathbf{F}' := \emptyset$
 6:    for each $f \in \mathbf{F}$ do
 7:       if $Y_i \in Scope(f)$ then
 8:          $\mathbf{F}' := \mathbf{F}' \cup \{f\}$
 9:          $\mathbf{F} := \mathbf{F} \setminus \{f\}$
10:       end if
11:    end for
12:    $f' := \bigoplus_{Y_i} \bigotimes_{f \in \mathbf{F}'} f$
13:    $F := F \cup \{f'\}$
14:    for each $X \in Scope(f')$ do
15:       if REVISE(X, f') then
16:          $\mathbf{P} := GGAC(\mathbf{P})$
17:          Break loop
18:       end if
19:    end for
20: end for
21: Return $g_{CBI}(\mathbf{Z}) := \bigotimes_{f \in \mathbf{F}} f$
```

Figure 3: Generalization of Variable Elimination with Arc Consistency Algorithm GVE-AC($\mathbf{P}, \mathbf{Z}$)

get a series of equivalent smaller CBI problems. The generalized arc consistency enforcement provides opportunities for performing inference more efficiently. We may either preprocess the CBI problem then apply regular systematic or stochastic inference approaches or simplify the problem during the application of inference approaches. For example, it is straightforward to incorporate generalized arc consistency enforcing into a generalized variable elimination algorithm [Chang, 2005], as shown in Figure 3, if a CBI problem $\mathbf{P} = (\mathbf{X}, \mathbf{D}, \mathbf{S}, \mathbf{F})$ has an eliminative commutative semiring $\mathbf{S}$.

4.2 Approximate Local Consistency and Monotonic Property

Given a CBI problem $\mathbf{P} = (\mathbf{X}, \mathbf{D}, \mathbf{S}, \mathbf{F})$, if the commutative semiring $\mathbf{S} = \langle \mathbf{A}, \oplus, \otimes \rangle$ is both eliminative and monotonic, we can propose a scheme to enforce local consistency approximately for this CBI problem. In other words, for an eliminative and monotonic commutative semiring, we use an element $\epsilon \in \mathbf{A}$ to approximate the multiplicative absorbing element $\alpha_{\otimes}$ that is equal to the additive identity element $\mathbf{0}$ for an eliminative commutative semiring.

Formally, we define the generalized ϵ arc consistency of a CBI problem as follows:

Definition 14 (A CBI Problem is ϵ-GGAC) *A CBI Problem $\mathbf{P} = (\mathbf{X}, \mathbf{D}, \mathbf{S}, \mathbf{F})$ with an eliminative commutative semiring $\mathbf{S}$ is ϵ generalized arc consistent (ϵ-GAC) if: $\forall f \in \mathbf{F}$, $\forall X \in Scope(f)$, $\forall x \in \mathbf{D}_X$, $\exists \mathbf{w}$, a value assignment of variables $Scope(f)_{-X}$, s.t. $f(x, \mathbf{w}) \geq_{\mathbf{S}} \epsilon$*

```
Input: A variable $X \in \mathbf{X}$, a constraint $f \in \mathbf{F}$, an element
    $\epsilon \in \mathbf{A}$
Output: TRUE if a value is removed from the domain of $X$;
    FALSE if else
 1: $flag := TRUE$
 2: for each $x \in \mathbf{D}_X$ do
 3:     for each value assignment w of $Scope(f)_{-X}$ do
 4:         if $\epsilon \leq_{\mathbf{S}} f(x, \mathbf{w})$ then
 5:             $flag := FALSE$
 6:             Break loop
 7:         end if
 8:     end for
 9:     if $flag$ then
10:         Remove $x$ from $\mathbf{D}_X$
11:         Return TRUE
12:     end if
13: end for
14: Return FALSE
```

Figure 4: Procedure ϵ-REVISE(X, f, ϵ) for Eliminating a Domain Value from a Variable According to the Approximation of a Local Constraint

An ϵ-GGAC algorithm can be achieved by modifying the REVISE procedure in Figure 2 to the procedure ϵ-REVISE(X, f, ϵ) in Figure 4. It is straightforward to show that the GGAC algorithm in Figure 1 with the procedure ϵ-REVISE leads to a ϵ-GGAC CBI problem of the given CBI Problem $\mathbf{P} = (\mathbf{X}, \mathbf{D}, \mathbf{S}, \mathbf{F})$ with an eliminative and monotonic commutative semiring $\mathbf{S}$.

Theorem 6 show that ϵ-GGAC algorithm can be used to simplify the original CBI problem with a controlled threshold. The simplified CBI problem gives a lower bound on the estimation of the inference task.

Theorem 6 (Lower Bound Estimation of ϵ-GGAC) *Given a CBI problem $\mathbf{P} = (\mathbf{X}, \mathbf{D}, \mathbf{S}, \mathbf{F})$ with an eliminative and monotonic commutative semiring $\mathbf{S} = \langle \mathbf{A}, \oplus, \otimes \rangle$, the ϵ-GGAC algorithm yields a CBI problem $\mathbf{P}' = (\mathbf{X}, \mathbf{D}', \mathbf{S}, \mathbf{F}')$ that is an approximation of $\mathbf{P}$ w.r.t. the results of their inference tasks. For any value assignment $\mathbf{z}$ of interested variables $\mathbf{Z}$, the inference task of $\mathbf{P}'$, $g'_{CBI}(\mathbf{z})$, is a lower bound of $g_{CBI}(\mathbf{z})$, w.r.t. the partial order $\leq_{\mathbf{S}}$ of the monotonic commutative semiring $\mathbf{S}$.*

Proof: Let (X, f) be a pair that is revised by the procedure ϵ-REVISE, where $x \in \mathbf{D}_X$ is removed because of ϵ generalized arc inconsistency. Consider the global constraint g of the combination of all the constraints in $\mathbf{P} = (\mathbf{X}, \mathbf{D}, \mathbf{S}, \mathbf{F})$. We have $g(\mathbf{X}) = f(\mathbf{X}_{\downarrow Scope(f)}) \otimes \bigotimes_{h \in \mathbf{F}, h \neq f} h(\mathbf{X}_{\downarrow Scope(h)})$. More specifically, for any value assignment $\mathbf{u}$ of variables $\mathbf{X}_{-X}$, we have $g(x, \mathbf{u}) = f(x, \mathbf{u}_{\downarrow Scope(f)}) \otimes \bigotimes_{h \in \mathbf{F}, h \neq f} h(\mathbf{u}_{\downarrow Scope(h)})$. Since $\alpha_{\otimes} \leq_{\mathbf{S}} f(x, \mathbf{u}_{\downarrow Scope(f)}) \leq_{\mathbf{S}} \epsilon$ and $\otimes$ is monotonic, we have $\alpha_{\otimes} \leq_{\mathbf{S}} g(x, \mathbf{u})$.

Given that $\alpha_{\otimes}$ is also the identity element of the operator $\oplus$ ($\mathbf{S}$ is eliminative) and $\oplus$ is monotonic, the inference task of $\mathbf{P}$ (Equation 1) is to compute:

$$
\begin{aligned}
g_{CBI}(\mathbf{Z}) &= \bigoplus_{\mathbf{Y}} g(X, \mathbf{X}_{-X}) \\
&= \bigoplus_{\mathbf{Y}} (g(X = x, \mathbf{X}_{-X}) \oplus g(X \neq x, \mathbf{X}_{-X})) \\
&\geq_{\mathbf{S}} \bigoplus_{\mathbf{Y}} g(X \neq x, \mathbf{X}_{-X}) \qquad (4)
\end{aligned}
$$

On the other hand, let us consider the global constraint g' of $\mathbf{P}' = (\mathbf{X}, \mathbf{D}', \mathbf{S}, \mathbf{F}')$. We have: $g'(\mathbf{X}) = \bigotimes_{f' \in \mathbf{F}'} f' = g(X \neq x, \mathbf{X}_{-X})$ according to the ϵ-GGAC algorithm. Then it is straightforward to get $g'_{CBI}(\mathbf{Z}) = \bigoplus_{\mathbf{Y}} g'(\mathbf{X}) \leq_{\mathbf{S}} g_{CBI}(\mathbf{Z})$ for every value assignment of interested variable subset $\mathbf{Z}$. □

Theorem 7 (Time Complexity of ϵ-GGAC) *The worst case time complexity of the ϵ-GGAC algorithm is $O(r \cdot d^{k+1})$, where r is the number of constraints, d is the maximum domain size, and k is the maximum scope size of constraints.*

Proof: The worst case time complexity of the ϵ-GGAC algorithm is the same as the GGAC algorithm, which is $O(r \cdot d^{k+1})$. □

5 Arc Consistency in Probability Assessment: An Example

Probability inference problems can be seen as constraint-based inference by treating conditional probability distributions (CPDs) as soft constraints over variables. A Bayesian network (BN) [Pearl, 1988] is a graphical representation for probability inference under conditions of uncertainty. BN is defined as a directed acyclic graph (DAG) where vertices $\mathbf{X} = \{X_1, \cdots, X_n\}$ denote *n* random variables and directed edges denote causal influences between variables. $\mathbf{D} = \{D_1, \cdots, D_n\}$ is a collection of finite domains for the variables. A set of conditional probability distributions $\mathbf{F} = \{f_1, \cdots, f_n\}$, where $f_i = P(X_i | Parents(X_i))$ is attached to each variable (vertex) X_i. Then the probability distribution over $\mathbf{X}$ is given by $P(\mathbf{X}) = \prod_{i=1}^{n} f_i$.

As a fundamental problem of probability inference, the probability assessment problem in Bayesian networks computes the posterior marginal probability of a subset of variables, given values for some variables as known evidence. We show in [Chang, 2005] that the probability assessment problem can be represented as a CBI problem using the commutative semiring $\mathbf{S_{prob}} = \langle \mathbb{R}^+ \cup \{0\}, +, \times \rangle$. We show in this section that our GGAC and ϵ-GGAC enforcing algorithms can preprocess the probability assessment problem efficiently. It is easy to show that $\alpha_{\otimes} = \mathbf{0} = 0$ and $\mathbf{S_{prob}}$ is monotonic.

The Bayesian network used here is the Insurance network from the Bayesian network Repository [Friedman *et al.*,]. The network has 27 variables and 27 non-binary constraints (CPDs). In our experiments, we randomly choose two variables as observed. The ϵ-GGAC algorithm is used to preprocess the problem. The junction tree algorithm in Lauritzen-Spiegelhalter architecture [Lauritzen and Spiegelhalter, 1988] is used to infer the marginal probability of every unobserved variable. We compare the number of binary operations required for probability assessment after using the ϵ-GGAC algorithm (shown as a fraction of the number required without ϵ-GGAC) and the resultant error of the marginal probability

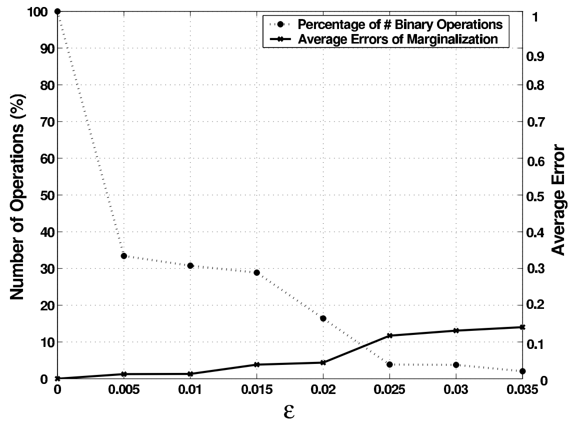


Figure 5: The number of binary operations required for probability assessment after using the ϵ-GGAC algorithm (shown as a fraction of the number required without ϵ-GGAC) and the resultant error of the marginal probability for the Insurance network as a function of ϵ

for the Insurance network as a function of ϵ in Figure 5. At each value of ϵ, we collect data for 5 runs. Results of our experiments are shown in Figure 5. It is clear that ϵ controls the tradeoff of the precision and the speed of the inference.

6 Conclusion and Future Works

As the most important local consistency techniques in constraint programming, arc consistency [Mackworth., 1977a] and its non-binary version, generalized arc consistency [Mackworth, 1977b; Mohr and Masini, 1988], are widely studied. The soft arc consistency algorithms [Schiex, 2000; Cooper and Schiex, 2004; Bistarelli, 2004] in the Semiring CSP [Bistarelli *et al.*, 1997] and Valued CSP [Schiex *et al.*, 1995] frameworks extend successfully the notion of arc consistency to the soft constraint processing. As the first result of this paper, we propose a weaker condition of applying generalized arc consistency enforcing techniques to a broader coverage of constraint-based inference problems, based on a semiring-based unified framework for CBI problems [Chang, 2005]. The weaker condition proposed here depends only on the existence and property of the combination absorbing element and does not depend on other semiring properties. We also present a concept of ϵ-GGAC that simplifies the representation of a CBI problem approximately. We show in this paper that the approximate inference task is a lower bound of the exact one w.r.t the total ordering of values in the commutative semiring structures. We also presented several generalized arc consistency enforcing algorithms in this paper. The worst time complexity of our generalization of generalized arc consistency enforcing algorithm is $O(r \cdot d^{k+1})$, where r is the number of constraints, d is the maximum domain size, and k is the maximum scope size of constraints. Our generalization of generalized arc consistency provides opportunities to researchers in the constraint programming community to extend their knowledge of local consistency enforcing approaches to other constraint-based inference problems such as probability inference and decoding problems.

Recently, many stronger local consistencies, such as directional arc consistency [Cooper and Schiex, 2004], full directional arc consistency [Larrosa and Schiex, 2003] and existential arc consistency [de Givry *et al.*, 2005], as well as Soft Arc Consistency [Cooper and Schiex, 2004] have been studied to solve Weighted CSP, Max-SAT, and Bayesian networks [Larrosa *et al.*, 2005]. We intend to compare the GGAC and ϵ-GGAC with these stronger local consistencies in handling different probability assessment problems in future work. Theoretical analysis of error bounds introduced by ϵ-GGAC algorithm is another research direction following the results of this paper.

Acknowledgments

We thank the anonymous reviewers for their comments on this paper. This research was supported by the Natural Sciences and Engineering Research Council of Canada and the Institute for Robotics and Intelligent Systems. Alan K. Mackworth holds a Canada Research Chair.

References

[Aji and McEliece, 2000] Srinivas M. Aji and Robert J. McEliece. The generalized distributive law. *IEEE Transactions on Information Theory*, 46:325–343, 2000.

[Bistarelli *et al.*, 1997] Stefano Bistarelli, Ugo Montanari, and Francesca Rossi. Semiring-based constraint satisfaction and optimization. *J. ACM*, 44(2):201–236, 1997.

[Bistarelli *et al.*, 2002] Stefano Bistarelli, Philippe Codognet, and Francesca Rossi. Abstracting soft constraints: framework, properties, examples. *Artif. Intell.*, 139(2):175–211, 2002.

[Bistarelli, 2004] Stefano Bistarelli. *Semirings for Soft Constraint Solving and Programming*. Springer-Verlag, 2004.

[Chang, 2005] Le Chang. Generalized constraint-based inference. Master's thesis, Dept. of Computer Science, Univ. of British Columbia, 2005.

[Cooper and Schiex, 2004] Martin Cooper and Thomas Schiex. Arc consistency for soft constraints. *Artificial Intelligence*, 154(1-2):199–227, 2004.

[de Givry *et al.*, 1997] Simon de Givry, Gérard Verfaillie, and Thomas Schiex. Bounding the optimum of constraint optimization problems. In *Proceedings of CP97*, pages 405–419, 1997.

[de Givry *et al.*, 2005] Simon de Givry, Matthias Zytnicki, Federico Heras, and Javier Larrosa. Existential arc consistency: Getting closer to full arc consistency in weighted csps. In *Proceedings of IJCAI-05*, Edinburgh, Scotland, 2005.

[Dechter, 1996] Rina Dechter. Bucket elimination: A unifying framework for probabilistic inference. In *12th Conf. on Uncertainty in Artificial Intelligence*, pages 211–219, 1996.

[Friedman *et al.*,] N. Friedman, M. Goldszmidt, D. Heckerman, and S. Russell. Bayesian network repository, http://www.cs.huji.ac.il/labs/compbio/repository/.

[Kask *et al.*, 2003] Kalev Kask, Rina Dechter, and Javier Larrosa. Unifying cluster-tree decompositions for automated reasoning. Submitted to the AIJ, June 2003.

[Kohlas and Shenoy, 2000] J. Kohlas and P.P. Shenoy. Computation in valuation algebras. In *Handbook of Defeasible Reasoning and Uncertainty Management Systems, Volume 5: Algorithms for Uncertainty and Defeasible Reasoning*, pages 5–40. Kluwer, Dordrecht, 2000.

[Larrosa and Schiex, 2003] Javier Larrosa and Thomas Schiex. In the quest of the best form of local consistency for weighted csp. In *Proceedings of IJCAI-03*, pages 239–244, Acapulco, Mexico, 2003.

[Larrosa *et al.*, 2005] Javier Larrosa, Thomas Schiex, Federico Heras, and Simon de Givry. Toolbar: a generic solver for WCSP, Max-SAT, and Bayesian networks, http://www.inra.fr/bia/t/degivry/toolbar.pdf, 2005.

[Lauritzen and Spiegelhalter, 1988] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society, Series B*, 50:157–224, 1988.

[Mackworth., 1977a] Alan K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.

[Mackworth, 1977b] Alan K. Mackworth. On reading sketch maps. In *IJCAI77*, pages 598–606, 1977.

[Mohr and Masini, 1988] Roger Mohr and G. Masini. Good old discrete relaxation. In *European Conference on Artificial Intelligence*, pages 651–656, 1988.

[Pearl, 1988] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann Publishers Inc., 1988.

[Rina and David, 2001] Dechter Rina and Larkin David. Hybrid processing of beliefs and constraints. In *Proceedings of the 17th Annual Conference on Uncertainty in Artificial Intelligence (UAI-01)*, pages 112–119, San Francisco, CA, 2001. Morgan Kaufmann Publishers.

[Schiex *et al.*, 1995] Thomas Schiex, Hélène Fargier, and Gerard Verfaillie. Valued constraint satisfaction problems: Hard and easy problems. In *IJCAI95*, pages 631–637, Montreal, 1995.

[Schiex, 2000] Thomas Schiex. Arc consistency for soft constraints. In *Proceedings of CP2000*, pages 411–424, 2000.

On the relation among Open, Interactive and Dynamic CSP[*]

Santiago Macho-Gonzalez and **Pedro Meseguer**
Institut d'Investigació en Intel.ligència Artificial, CSIC
Campus UAB, 08193 Bellaterra, Spain
{smacho,pedro}@iiia.csic.es

Abstract

In previous work, the notions of Open, Interactive and Dynamic CSP have been independently defined. *Open* constraint satisfaction is a new model where values are incrementally gathered during problem solving. Domains are assumed unbounded. *Interactive* constraint satisfaction also deals with partially known domains, assuming implicitly that domains are finite. *Dynamic* constraint satisfaction deals with problems of dynamic nature (as configuration design or model composition) where variables, domains and constraints are subject to frequent changes. In this paper, we study the relationship between these three models, showing that Interactive CSP can be seen as a particular case of Open and Dynamic. We have applied two algorithms, FOCSP (developed for Open) and LC (developed for Dynamic) to solve Interactive CSP. We provide experimental results of this evaluation.

1 Introduction

With the increasing use of Internet, many problem-solving tasks such as resource allocation, scheduling, planning, and configuration pose themselves in an *open* setting involving multiple participants. Existing search-based problem-solving techniques are based on the closed-world assumption and require that all options be collected before problem-solving starts. The approach of turning the web into a virtual database often leads to gathering more information than needed to solve the problem.

In previous work [Faltings and Macho-Gonzalez, 2005; 2002], a new model called Open CSP was defined, as a way to integrate information gathering and problem solving. This model starts solving a CSP from a state where all domains are possibly empty, and it dynamically asks for values while the CSP has not been solved. This process stops as soon as a solution is found, optimising the number of values queried to find the solution.

With a similar motivation, the Interactive CSP model was proposed [**?**], to deal with problems with partially defined domains. It is implicitly assumed that the domains are finite, while in Open CSP domains remain unbounded. As consequence, Interactive CSP appears to be included in the Open CSP framework.

In a dynamic environment, tasks usually change. As a consequence, CSPs that represent these tasks evolve, and variables, domains and constraints may change over time. The Dynamic CSP model [Dechter and Dechter, 1988] was defined to solve CSP in such dynamic environments.

This paper brings a review of the Open, Interactive and Dynamic CSP approaches and their relationship. In the next sections, we will see that an Interactive CSP can be seen as a particular case of Open CSP. In addition, Interactive CSP can be seen as a particular case of Dynamic CSP. Therefore, algorithms developed for Open or Dynamic models could be used for solving Interactive CSP.

The remain of this paper is structured as follows. In Section 2 we give a brief description of the Open CSP model, including the FOCSP solving algorithm. In Section 3 we give a brief description of the Interactive CSP model, showing that it can be seen as a particular case of Open CSP. Section 4 contains an overview of the Dynamic CSP approach. In Section 5 we indicate the similarities and differences between them, showing that an Interactive CSP can be seen as a Dynamic CSP. In Section 6, we present the LC algorithm, initially developed for Dynamic CSP, that now can be applied to Interactive CSP. Experimental results on the use of FOCSP and LC algorithms on Interactive CSP instances appear in Section 7. Finally, Section 8 contains some conclusions and lines for further research.

2 Open CSP

There are real problems which are difficult to solve with the classical CSP approach of collecting all values before solving the problem. For example in configuration, such as configuring a PC or configuring a trip. On the web there are may data sources that each apply values for each component. Query all them is not feasible, we are interested just in querying until a solution is found. Other examples are when the number of options can become infinitely large if they are generated on demand, for example if you configure an investment portfolio you can get certain investment products in an unbounded number of varieties. Composing web services, configuring a supply chain (asking for offers from suppliers until you have a

[*]Partially supported by the Spanish REPLI project TIC-2002-04470-C03-03.

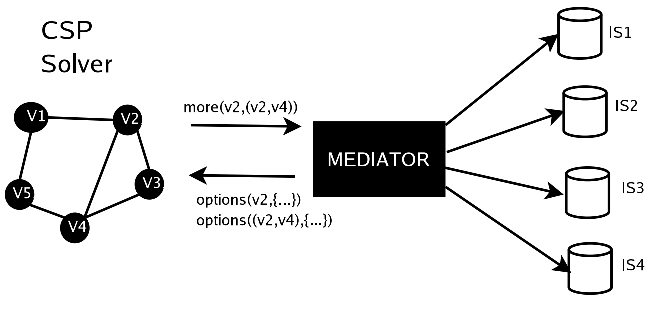


Figure 1: *Elements of an open constraint satisfaction problem*

consistent plan),..... are more examples of problems for which the classical CSP approach is unfeasible.

To address these problems, the Open CSP approach [Faltings and Macho-Gonzalez, 2005; 2002] was proposed. In Figure 1 we show the important elements that occur in an open setting. The problem-solving process is modelled abstractly as the solution of a constraint satisfaction problem. The choices that make up domains and permitted tuples of the CSP are distributed throughout an unbounded network of information servers $IS_1, IS_2, ...$, and accessed through a mediator ([Wiederhold and Genesereth, 1997]).

More precisely, an *Open constraint satisfaction problem* (Open CSP)[Faltings and Macho-Gonzalez, 2005; 2002] is a possibly infinite sequence $\langle$CSP(0), CSP(1), $\ldots\rangle$ of CSP instances. An instance CSP(i) is the tuple $\langle V, D(i), C(i)\rangle$ where,

- $V = \{v_1, v_2, ..., v_n\}$ is a set of n variables.
- $D(i) = \{D_1(i), D_2(i), ..., D_n(i)\}$ is the set of domains for CSP(i) where variable v_k takes values in $D_k(i)$. Domains grow monotonically with i, $D_k(i) \subseteq D_k(i+1)$ for all k.
- $C(i) = \{c_1(i), c_2(i), \ldots, c_r(i)\}$ is a set of r constraints. A constraint $c(i)$ involves a sequence of variables $var(c(i)) = \langle v_p, \ldots, v_q\rangle$ denominated its scope. The extension of $c(i)$ is the relation $rel(c(i))$ defined on $var(c(i))$, formed by the permitted value tuples on the constraint scope. Relations grow monotonically, $rel(c_k(i)) \subseteq rel(c_k(i+1))$ for all k.

A *solution* is a set of value assignments involving all variables such that for some i, each value belongs to the corresponding domain in $D(i)$ and all value combinations are allowed by the constraints $C(i)$ of $CSP(i)$. Solving an Open CSP requires an integration of search and information gathering. It starts from a state where all domains are empty, and the first action is to find values that fill the domains and allow the search to start. As long as the available information does not include enough values to make the CSP solvable, the problem solver initiates further information gathering requests to obtain additional values. The process stops as soon as a solution is found.

In [Faltings and Macho-Gonzalez, 2002] it was developed the FOCSP algorithm for solving Open CSP. It appears in Figure 2. It is based on the idea that we gather new values only when the current instance $CSP(i)$ has no solution. In that case, it usually contains a smaller subproblem that already has no solution, and $CSP(i)$ can be made solvable only by creating a solution to that subproblem. Information gathering thus should focus on the variables in the subproblem, as follows. Let x_i be the deepest variable that backtracking has reached when trying to solve $CSP(i)$, following a static variable ordering. x_i is called the *failed variable*, and it is proved that it belongs to the smallest subproblem without solution inside $CSP(i)$ [Faltings and Macho-Gonzalez, 2002]. After a failed search, an additional value is requested for that variable, that now is considered first in the static variable ordering and the search restarts.

```
function FOCSP$(X, D, C)$
  $i \leftarrow 1, k \leftarrow 1$
  repeat {backtrack search}
    if $exhausted(d_i)$ {backtrack} then
      $reset-values(d_i), i \leftarrow i-1$
    else
      $k \leftarrow max(k,i), x_i \leftarrow nextvalue(d_i)$
      if $consistent(\{x_1..x_i\})$ then $i \leftarrow i+1$
      if $i > n$ then return $\{x_1, ..., x_n\}$ as a solution;
  until $i = 0$
  if $exhausted(d_k)$
    if $(\forall i \in 1..k-1) exhausted(d_i)$ then return failure
    else
      $nv \leftarrow more(x_k)$
      if $nv \neq nomore$ then $d_k \leftarrow nv \cup d_k$
  reorder variables so that $x_k$ becomes $x_1$ (relative order of
    others remains the same)
  return FOCSP(X,D,C)
```

Figure 2: The FOCSP algorithm.

In [Faltings and Macho-Gonzalez, 2002], it is shown that if the current instance $CSP(i)$ contains a minimal unsolvable subproblem, the FOCSP algorithm will either solve it or fall into a mode where it cycles through the variables in this minimal subproblem and gathers values for each of them. This allows showing that the algorithm is complete even in the presence of unbounded domains.

3 Interactive CSP

Very related with Open CSP is the Interactive CSP model introduced by [**?**]. An Interactive CSP (ICSP) has partially known domains for its variables. When solving an Interactive CSP, new values are requested, until finding a solution or proving that no solution exists. It is implicitly assumed that variable domains are finite.

More precisely, an Interactive CSP is defined as a triple (V, D, C), where V is a set of n variables, D is a collection of domains and C is a set of constraints. The new elements to consider in the Interactive CSP model are interactive domains and constraints. An *interactive domain* D_i (the domain of variable v_i) has two parts, $D_i = Known_i \cup Unknown_i$; $Known_i$ is the set of available values for v_i, while $Unknown$ as the set of not yet available values for v_i. An *interactive constraint* is a constraint including in its scope one or more variables with interactive domains, so not all value combinations are known. As soon as new values are known for these variables, new constraint tuples are added to the constraint.

Formally, an Interactive CSP can be seen as a particular case of Open CSP, where values for interactive domains are obtained during the solving process. In this sense, we could define Interactive CSP as a finite sequence $\langle$CSP(0), CSP(1), $\ldots\rangle$ of CSP instances. An instance CSP(i) is the tuple $\langle V, D(i), C(i)\rangle$ defined as in the Open CSP case. We stress two properties here: $D_k(i) \subseteq D_k(i+1)$ for all k and $rel(c_k(i)) \subseteq rel(c_k(i+1))$ for all k.

The main difference between Interactive and Open CSP models consists on the assumed domain size. In Open CSP domains are unbounded, so the sequence of CSP instances may be infinite. This is not the case for Interactive CSP, where domains are assumed finite so the sequence of CSP instances is necessarily finite.

In the Interactive CSP model, it is possible to use heuristically some of the known constraints to guide the acquisition of new values. This feature depends on the concrete application to solve, but no specific condition is requested on the basic model. In addition, some examples suggest the use of a high-level mediator, able to acquire all values consistent with a specific assignment. This feature could be seen as a way to implement the value acquisition process.

Some specific solving algorithms have been proposed for this model. The forward checking algorithm is modified so that when domains become empty, it launches a specific request for additional values that would satisfy the constraint on that variable. In earlier work([Cucchiara *et al.*, 1997]), the same authors also show how arc consistency algorithms can be adapted with the right dependency structures so that consistency can be adapted to values that might be added later. Interactive CSP uses constraint propagation to reduce the search space by pruning the assignments which cannot appear in any consistent solution and to guide the search by generating new constraints at each step[Cucchiara *et al.*, 1997]. An interesting application for ICSP is defined in [Cucchiara *et al.*, 1997] where ICSP is used for object recognition and identification in a visual system.

4 Dynamic CSP

As mentioned before, many interesting problems in Artificial Intelligence such as scheduling, planning and configuration can be modelled as CSP. But in many situations there is not complete knowledge of the environment at early stage of the problem; variables and constraints could evolve over the time.

A *Dynamic constraint satisfaction problem* (Dynamic CSP) [Bellicha, 1993; Bessière, 1991; 1992; Dechter and Dechter, 1988] is a finite sequence $\langle$CSP(0), CSP(1),$\ldots\rangle$ of CSP instances, where each CSP(i) differs from the previous one by the addition or removal of some constraints. It is easy to see that all possible changes of a CSP can be expressed in terms of constraint additions or removals. Implicitly, it is usually assumed that (i) changes between consecutive instances are local, that is, they do not affect the whole CSP, and (ii) solutions of consecutive instances are not very different, they differ in a few number of values.

Solving a Dynamic CSP implies solving each instance of the sequence. The first instance is solved from scratch, and it is always possible to apply this method to any subsequent one. However, this approach presents two drawbacks,

- Inefficiency. Solving an instance from scratch could repeat much of the work done to solve previous instances, which may be unacceptable for some applications.
- Instability. Solving instances from scratch may result in the fact that successive solutions are far or unrelated. This may be unpleasant or undesired if some kind of continuity among solutions is required.

To avoid these drawbacks, when solving an instance we aim at reusing as much as possible the solving episodes of previous instances. Obviously, the difficult case appears when constraints are added between consecutive instances, since adding a new constraint could invalidate the previous solution. Removing constraints between consecutive instances does not cause any problem, because the previous solution is still valid.

Several algorithms were developed for solving Dynamic CSPs. They can be divided into two groups:

- *Remember what has been discovered (Recording no-goods)*. This method is based on recording any no-good and its justification, in order to reuse it in the framework of any new CSP (with a new constraint or without a constraint) to prevent failures. The method is explained in [Schiex and Verfaillie, 1994; Jiang *et al.*, 1994].
- *Local repair methods*. This method starts from any previous consistent assignment (that could be a solution of the former CSP) and repairing it, using a sequence of local modifications. The method is based on the characteristics of a dynamic environment, exposed before. An algorithm based on this method can be found on [Verfaillie and Schiex, 1994].

5 Interactive CSP as Dynamic CSP

An Interactive CSP can be seen as a particular case of Dynamic CSP, as follows. In Interactive CSP, the operation that passes from a problem instance to the next one is *acquire value*, getting a new value for a particular variable. Then, the variable domain is extended with that value, and the relational part of constraints involving such variable are enlarged with the allowed tuples that contain the new value. This process can be modelled in Dynamic CSP as follows. Adding a new value is equivalent to removing a unary constraint which disallowed this value in the domain of the corresponding variable, so that value is now available. Enlarging the constraints in which the variable is involved is equivalent to replacing (removing plus adding) the previous constraints by the enlarged ones. This simple idea is depicted in Table 1.

Interactive CSP	**Dynamic CSP**
Acquire $value$	Remove unary constraint that forbids $value$ Replace (remove and add) some constraints

Table 1: Interactive CSP vs Dynamic CSP.

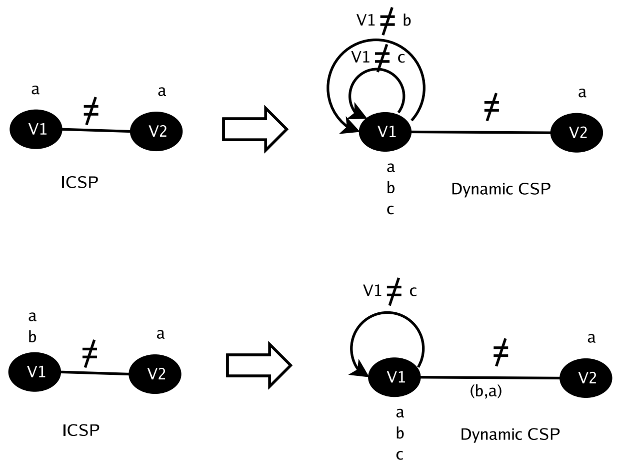


Figure 3: *From an Interactive CSP to a Dynamic CSP*

An example of this reformulation appears in Figure 3. Let us suppose that an Interactive CSP has two variables $\{V1, V2\}$ such that $D1 = \{a, b, c\}$ and $D2 = \{a\}$. There is an inequality constraint between them. If value a is known for $V1$ and $V2$, the problem can be transformed into a Dynamic CSP with the same variables, two new unary constraints (one for each remaining value of $V1$, $V1 \neq b, V1 \neq c$), and the inequality constraint. In this example, acquiring a new value b for the variable $V1$ of the Interactive CSP is equivalent to relax the unary constraint $V1 = b$ in the Dynamic CSP, and adding the pair (b, a) as permitted in the binary constraint.

At first glance one may think that this approach requires to know all the values of the domains from the beginning, to form the variable domains of the Dynamic CSP. However, this is not the case. It is enough to know the maximum number of values for each variable, say d_i for v_i. Initially, the problem state is as follows. The domain of v_i is a set of d_i dummy values $\{dummy_1, \ldots, dummy_{d_i}\}$. When value a is found, it replaces a dummy value, say $dummy_1$, in the variable domain (that now becomes $\{a, dummy_2, \ldots, dummy_{d_i}\}$), and in the constraints. At this point, the changes in the constraints mentioned in Table 1 are performed.

Strictly speaking, this model is an extension of the standard model of Dynamic CSP, where all domains are known from the beginning. The existence of dummy values which are replaced by real values as search progresses is not a big issue for the standard Dynamic CSP model, because the domain size does not change, and dummy values are replaced by real ones only once. This is the only extension that the standard Dynamic CSP model requires to include Interactive CSP. We call this new model the *Extended Dynamic CSP*. The relation between these models appears in Figure 4.

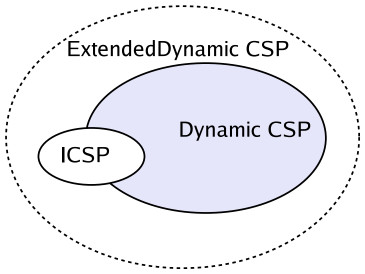


Figure 4: *Relation between Interactive CSP and Dynamic CSP*

Now, we can apply Dynamic CSP algorithms to solve Interactive CSP problems. In the next Section, we take the local changes algorithm [Verfaillie and Schiex, 1994], a specific algorithm for Dynamic CSP, and we apply it for Interactive CSP solving, obtaining interesting results.

6 The LC algorithm

We have applied the local changes (LC) algorithm [Verfaillie and Schiex, 1994], originally developed for solving Dynamic CSP, to solve Interactive CSP instances. The LC algorithm appears in Figure 5.

This algorithm is based on the following idea: instead of solving from scratch the Interactive CSP every time a new value is added (i.e. to $v = nv$) during the solving process (as described in the FOCSP algorithm), it is possible to solve the Interactive CSP removing all variables whose assignment is incompatible with $v = nv$ and entering again these variables one after another without modifying the assignment $v = nv$. The basic idea is to extend the compatible assignment $v = nv$ and the compatible variables, into an assignment which involves all variables.

The LC algorithm of Figure 5 works as follows. V_1 is the set of variables with an assignment which we will not modify, V_2 is the set of variables with an assignment we could modify and V_3 is the set of unassigned variables. The initial call to the algorithm is $LC(\emptyset, \emptyset, X)$ where X is the set of variables of the Interactive CSP. Function `lc-variables`(V_1, V_2, V_3) returns *true* if there exists a consistent assignment for all the variables, without modifying the assignment of V_1, and *false* otherwise. Function `lc-variable`(V_1, V_2, v, d) returns *true* if there is a consistent assignment for variables $V_1 \cup V_2 \cup \{v\}$ without modifying the assignment of V_1, and *false* otherwise. This function includes the query to get new values for a given variable, in the first four lines of the function. If the domain of v is exhausted (empty), a new value is requested through the call $more(v)$. If the returned value is $nomore$, it means that no more values are available for variable v. In that case, the function returns *false*. Otherwise, the new value enters in the variable domain and the process continues. Function `lc-value`(V_1, V_2, v, val) returns *true* if there is an assignment for variables in V_2 such that this assignment plus the assignment of V_1 plus (v, val) form a consistent assignment. To do this, this function explores the possible assignments for V_2, keeping V_1 and (v, val) fixed, through the call `lc-variables`$(V_1 \cup v, V2 - V_3, V_3)$.

Theorem. *Supposed that an Interactive CSP is solvable (resp. insolvable). Then, calling the LC algorithm with the statement $LC(\emptyset, \emptyset, X)$ (where X is the set of variables of the problem) returns a solution (resp. failure) of the Interactive CSP. Thus, the algorithm is complete.*

```
function lc$(X, D, C)$: boolean;
  return lc-variables$(\emptyset, \emptyset, X)$;

function lc-variables$(V_1, V_2, V_3)$: boolean;
  if $V_3 = \emptyset$ then return true;
  else
    $v \leftarrow select - var(V_3)$;
    $d \leftarrow domain(v)$;
    if lc-variable$(V_1, V_2, v, d)$
      then return lc-variables$(V_1, V_2 \cup \{v\}, V_3 - \{v\})$;
      else return false;

function lc-variable$(V_1, V_2, v, d_v)$: boolean;
  if $exhausted(d)$ then
    $nv \leftarrow more(v)$;
    if $nv = nomore$ then return false;
    else $d \leftarrow \{nv\}$;
  $v \leftarrow select - val(d)$;
  $save - assig(V_2)$;
  if lc-value$(V_1, V_2, v, val)$ then return true;
  else
    $restore - assig(V_2)$;
    return lc-variable$(V_1, V_2, v, d - \{val\})$;

function lc-value$(V_1, V_2, v, val)$: boolean;
  $A_1 \leftarrow assignment(V_1)$;
  $A_{12} \leftarrow assignment(V_1 \cup V_2)$;
  if $A_1 \cup \{(v, val\}$ is inconsistent then return false;
  else if $A_{12} \cup \{(v, val\}$ is consistent then return true;
  else
    $V_3 \leftarrow$ nonempty subset of $V_2$ such that
      $A_{123} \leftarrow assignment(V_1 \cup V_2 - V_3)$ and
      $A_{123} \cup \{(v, val\}$ is consistent
    $unassign - vars(V_3)$;
    return lc-variables$(V_1 \cup \{v\}, V_2 - V_3, V_3)$;
```

Figure 5: The LC algorithm.

Proof: The LC algorithm is described in [Verfaillie and Schiex, 1994], where the authors give a proof of its correctness, completeness and termination for known variable domains. Because the LC algorithm works with a subset of the complete variable domain (or the complete domain in the worst case), then the proof is still valid.

Figure 6 shows two examples of how the LC algorithm works. In figure 6(a), we have a CSP instance in an Interactive CSP without solution with the inconsistent assignment $V1 = b, V2 = c, V3 = a, V4 = a$. If we decide to add a new value to variable $V4$ then we remove all the assignments of the variables which are inconsistent with the new value i.e $V4 = b$. Next step is reassign these variables one after another without modifying the assignment. In figure 6(a) having the new assignment $V4 = b$, the set of compatible assignments is $A_C = \{V2 = b, V3 = a\}$ and the set of incompatible assignments is $A_{NC} = \{V1 = b\}$. Next step is to assign a new value for $V1$ which will be compatible with the assignment $A_C \cup \{V4 = b\}$[1]. Figure 6(b) shows another example. The CSP instance of the Interactive CSP has no solution with the inconsistent assignment $V1 = b, V2 = a, V3 = b$. If we decide to add a new value for variable $V3$, lets say $V3 = c$, this value is consistent with the previous assignment $A = \{V1 = b, V2 = a\}$, thus we have a solution.

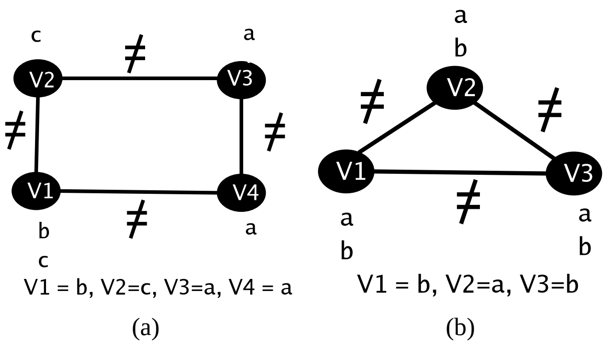


Figure 6: Examples of the LC algorithm.

7 Experimental Results

We compared the performance of the LC algorithm against the FOCSP algorithm described in [Faltings and Macho-Gonzalez, 2005; 2002]. The FOCSP is based on the idea of using a failed CSP search to determine for which variable additional values should be collected. The idea is that when a CSP has no solution, there exists an unsolvable subproblem that causes the inconsistency. If we want to make solvable the CSP we need to add a value to one of the variables of this unsolvable subproblem. We can identify a variable that belongs to this subproblem using a failed backtracking. This variable is called *failed variable*.

To compare the algorithms, we are interested in the number of checks needed to solve the Interactive CSP and the number of accesses to information sources until a solution is found. We generated 100000 random Interactive CSPs, with between 5 to 18 variables and 2 to 7 values per variable, with random constraints, forcing the graph to be at least connected and at most complete.

Figure 7 shows the number of checks against the number of variables, studying the performance of the algorithms when we increase the number of variables. It is shown that the LC algorithm has a much better performance that the FOCSP algorithm. The FOCSP algorithm redoes again the same solving process every time a new value is added, while the LC algorithm uses the information from the previous assignments (compatible assignments, incompatible assignments) for solving without solving again the CSP from scratch.

Figure 8 shows the number of queries against the number of variables. We can see that LC and FOCSP algorithms have a much better performance than the classical approach

[1] In the example of figure 6(a) this step is not enough for solving the problem, because there is not any assignment of $V1$ compatible with $A1 \cup \{V4 = b\}$. The algorithm will decide to repeat the process adding a new value to variable $V1$

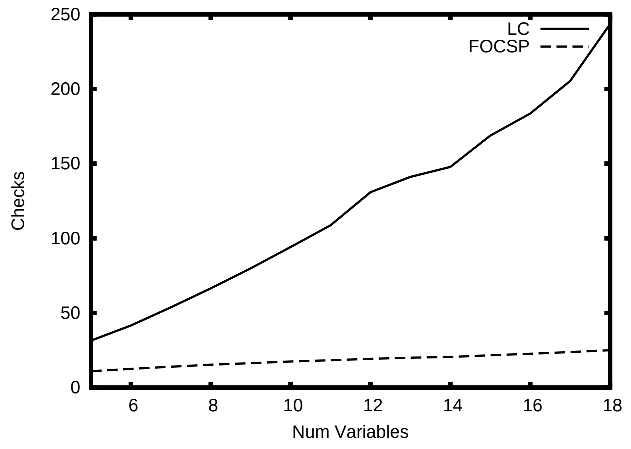


Figure 7: *Comparison of the number of checks against the number of variables*

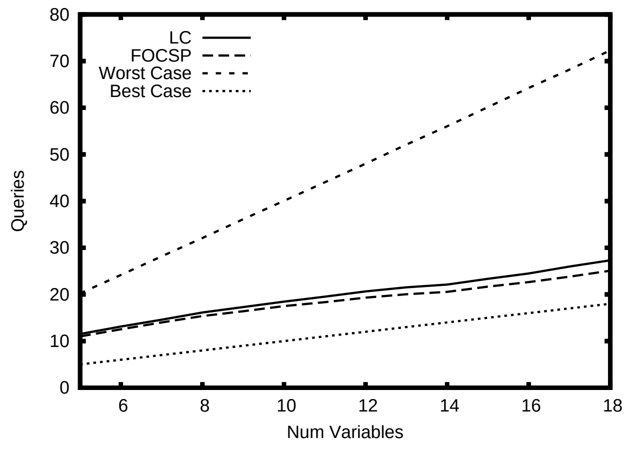


Figure 8: *Comparison of the number of queries against the number of variables*

of collecting all values and solve the problem. Both, LC and FOCSP query nearly the same number of values to find a solution.

Figures 7 and 8 show the improvements of the LC algorithm described in the previous section. Empirically it is shown in figure 7 that reuse previous work on failed branches is more perfomant on average than detect the failed variable and redo the problem as the FOCSP algorithm does.

It is also interesting to analyze the fact that the number of queries of LC algorithm is always slightly higher than the number of queries of FOCSP. This may be related with the way values are queried by both algorithms. While consecutive queries of FOCSP ask for values of different variables, consecutive queries of LC may ask the complete domain of a variable. Therefore, in some cases LC may ask more than needed to find a solution. This point is subject to current research.

8 Conclusions

In this paper we have analyzed the relation among Open, Interactive and Dynamic CSP. These models, different from the classical CSP, have appeared in different moments motivated by different applications. We have shown that Interactive CSP can be seen as a particular class of Open CSP (restricted to finite domains). In addition, we have also shown that Interactive CSP can be seen as a particular class of Dynamic CSP (strictly speaking, a class of extended Dynamic CSP). As consequence, algorithms used to solve Open CSP and Dynamic CSP can be used to solve Open CSP. Based on this relationship, we have applied the FOCSP algorithm, (initially developed for Open CSP) and the LC algorithm algorithm (initially developed for Dynamic CSP), to solve Interactive CSP instances. We have found that the LC algorithm reduces dramatically the number of checks with respect to FOCSP, just slightly increasing the number of queries needed to find a solution.

We think that this relationship between Open, Interactive and Dynamic CSP is a promising avenue for research, that we will further investigate in the near future.

References

[Bellicha, 1993] Amit Bellicha. Maintenance of a solution in a dynamic constraint satisfaction problem. *Applications of Artificial Intelligence in Engineering*, pages 261–274, 1993.

[Bessière, 1991] Christian Bessière. Arc-consistency in dynamic constraint satisfaction problems. In *Proceeding of the Ninth National Conference on Artificial Intelligence*, pages 221–226, 1991.

[Bessière, 1992] Christian Bessière. Arc-consistency for non-binary dynamic constraint satisfaction problems. In B. Neumann, editor, *Proceedings of the 10th European Conference on Artificial Intelligence*. John Wiley & Sons, Ltd, 1992.

[Cucchiara *et al.*, 1997] Rita Cucchiara, Evelina Lamma, Paola Mello, and Michela Milano. An interactive constraint-based system for selective attention in visual search. In *International Symposium on Methodologies for Intelligent Systems*, pages 431–440, 1997.

[Dechter and Dechter, 1988] Rina Dechter and Avi Dechter. Belief maintenance in dynamic constraint networks. In *Proceedings of the Seventh Annual Conference of the American Association of Artificial Intelligence*, pages 37–42, 1988.

[Faltings and Macho-Gonzalez, 2002] Boi Faltings and Santiago Macho-Gonzalez. Open constraint satisfaction. *CP-2002*, pages 356–370, 2002.

[Faltings and Macho-Gonzalez, 2003a] Boi Faltings and Santiago Macho-Gonzalez. Incentive compatible open constraint optimization. In *Proceedings of AAMAS 2003*, July 2003.

[Faltings and Macho-Gonzalez, 2003b] Boi Faltings and Santiago Macho-Gonzalez. Open constraint optimization. In *Proceedings of the 9th International Conference on*

Principles and Practice of Constraint Programming (CP-2003), Lecture Notes in Computer Science. Springer, September 2003.

[Faltings and Macho-Gonzalez, 2005] Boi Faltings and Santiago Macho-Gonzalez. Open constraint programming. *Artificial Intelligence*, 161:181–208, 2005.

[Jiang *et al.*, 1994] Yuejun Jiang, Thomas Richards, and Barry Richards. No-good backmarking with min-conflicts repair in constraint satisfaction and optimization. In *Proceedings of Principles and Practice of Constraint Programming 94; reprinted in Principles and Practice of Constraint Programming 94*, pages 21–39, 1994.

[Lamma *et al.*, 1999] Evelina Lamma, Paola Mello, Michela Milano, Rita Cucchiara, Marco Gavanelli, and Massimo Piccardi. Constraint propagation and value acquisition: Why we should do it interactively. In *IJCAI*, pages 468–477, 1999.

[Schiex and Verfaillie, 1994] Thomas Schiex and Gérard Verfaillie. Nogood recording for static and dynamic constraint satisfaction problems. *International Journal on Artificial Intelligence Tools*, 3(2):187–207, 1994.

[Verfaillie and Schiex, 1994] Gérard Verfaillie and Thomas Schiex. Solution reuse in dynamic constraint satisfaction problems. In *Proceedings of the Twelfth Conference of the American Association of Artificial Intelligence*, pages 307–312, 1994.

[Wiederhold and Genesereth, 1997] Gio Wiederhold and Michael R. Genesereth. The conceptual basis for mediation services. *IEEE Expert*, 12(5):38–47, 1997.